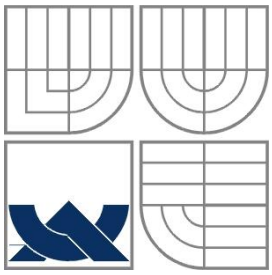
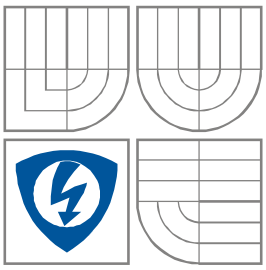


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF CONTROL AND INSTRUMENTATION

UNIVERZÁLNÍ ŘÍDICÍ SYSTÉM PRO QUADROCOPTER

UNIVERSAL CONTROL SYSTEM FOR QUADROCOPTER

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

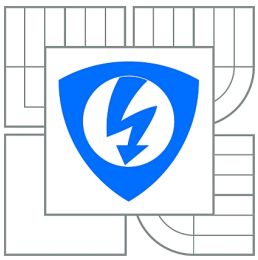
AUTOR PRÁCE
AUTHOR

Bc. Petr Gábrlík

VEDOUCÍ PRÁCE
SUPERVISOR

doc. Ing. Luděk Žalud, Ph.D.

BRNO 2011



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. Petr Gábrlík	***ID:***	111039
Ročník:	2	***Akademický rok:***	2011/2012

NÁZEV TÉMATU:

Univerzální řídicí systém pro quadrocopter

POKYNY PRO VYPRACOVÁNÍ:

Vyberte a seznamte se s vhodným mikrokontrolérem pro řídicí systém quadrocopteru. Navrhněte a realizujte řídicí program pro vybraný mikrokontrolér. Systém by měl umožnit připojení potřebných snímačů a periferií, konfiguraci základních parametrů quadrocopteru a komunikaci s nadřazeným operátorským systémem CASSANDRA-WPF. Realizujte základní algoritmus pro stabilizaci rotačních pohybů quadrocopteru. Podle pokynů vedoucího systém otestujte.

DOPORUČENÁ LITERATURA:

[1] HOLLAND, John. Designing Autonomous Mobile Robots. [s.l.] : Newnes, c2004. 335 s. ISBN 0-7506-7683-3.
[2] VÁŇA, Vladimír. ARM pro začátečníky. Praha : BEN - technická literatura, 2009. 195 s. ISBN 978-80-7300-246-6.
[3] HOROUT, Pavel. Učebnice jazyka C, KOPP, 2004, IV. přepracované vydání, ISBN 80-7232-220-6

Termín zadání:	6.2.2012	***Termín odevzdání:***	21.5.2012

Vedoucí práce: doc. Ing. Luděk Žalud, Ph.D.
Konzultanti diplomové práce:

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

UPOZORNĚNÍ:

Abstrakt

Cílem této diplomové práce je návrh a realizace univerzálního řídicího systému pro létající robot typu quadrocopter. Její první část se zabývá způsoby, jakými lze řešit program pro mikrokontrolér. Zvláštní pozornost je věnována operačnímu systému reálného času FreeRTOS, který je určen pro mikrokontroléry. Druhá část práce se zaměřuje na popis zvoleného řídicího mikrokontroléru LM3S8962 a celkového hardwarového řešení. Jedna kapitola je také věnována začlenění robotu do řídicího systému Cassandra-WPF a komunikaci s ním. Třetí část se zabývá identifikací nové konstrukce robotu, především akčních členů a vytvořením jednoduchého matematického modelu. Díky němu jsou navrženy regulátory pro stabilizaci náklonu a sklonu robotu, které jsou dále otestovány na reálné konstrukci robotu. Závěr práce je věnován tvorbě řídicí aplikace pro operační systém FreeRTOS, její otestování a začlenění stabilizačních algoritmů.

Klíčová slova

quadrocopter, UAV, Uranus, Cassandra-WPF, mikrokontrolér, LM3S8962, FreeRTOS

Abstract

The Thesis objective is the design and implementation of the universal control system for a flying robot, quadrotor concept. The first part deals with ways of solving program for microcontrollers. The special attention is given to the FreeRTOS real-time operating system, which is designed for microcontrollers. The second part of the Thesis is focused on the description of the chosen microcontroller LM3S8962 and the hardware solution. One chapter is devoted to the integration of the robot to the Cassandra-WPF robotic control system. The third part deals with the identification of the new robot construction and the mathematical model creation. Using the model controllers for stabilization pitch and roll are designed and their functionality is verified on a physical model. The last part of the Thesis is focused on the FreeRTOS implementation and the control application creation.

Keywords

quadrocopter, UAV, Uranus, Cassandra-WPF, microcontroller, LM3S8962, FreeRTOS

Bibliografická citace:

GÁBRLÍK, P. *Univerzální řídicí systém pro quadrocopter*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2012. 83s. Vedoucí diplomové práce byl doc. Ing. Luděk Žalud, Ph.D.

Prohlášení

„Prohlašuji, že svou diplomovou práci na téma Univerzální řídicí systém pro quadrocopter 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: **21. května 2012** ……
podpis autora

Poděkování

Děkuji vedoucímu diplomové práce doc. Ing. Luďkovi Žaludovi, Ph.D. za účinnou metodickou, pedagogickou a odbornou pomoc a další cenné rady při zpracování mé diplomové práce.

V Brně dne: **21. května 2012**

……
podpis autora

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

1 ÚVOD

Tato diplomová práce je součástí dlouhodobého vývoje robotického létajícího prostředku probíhajícího na Ústavu automatizace a měřící techniky Vysokého učení technického v Brně. Navazuje na dřívější poznatky, které již byly na ústavu publikovány formou bakalářských a diplomových prací a konferenčních příspěvků. Cílem celého projektu je vytvoření univerzální platformy více rotorového létajícího robotu kategorie UAV, který by v závislosti na konfiguraci dokázal plnit různorodé mise ve vnitřním a vnějším prostřední, s různou úrovní autonomnosti a dálkového řízení, s využitím různých navigačních a komunikačních prostředků a začleněním do stávajícího robotického systému, který je na ústavu vyvíjen. Díky komplexnosti projektu je důležité jeho členění na jednotlivé logické a funkční celky, které jsou vyvíjeny samostatně, ale v úzké spolupráci. Výsledkem této diplomové práce proto není realizace celého robotu, ale pouze jeho několika nezbytných součástí.

První verze quadrocopteru na ústavu vznikla jako výsledek trojice bakalářských prací, které si dávaly za úkol vytvoření funkčního experimentálního modelu. Práce [1] se zabývá programovou obsluhou a stabilizací, práce [2] návrhem hardwaru a komunikací a práce [3] navigačním systémem GPS. Výsledné programové, hardwarové a konstrukční řešení přineslo nespočet zkušeností a poznatků, z kterých tato práce čerpá. Robot vykazoval základní letové schopnosti při dálkovém řízení a celá konstrukce, znázorněná na Obrázku 1, může sloužit k dalším experimentům. Dvojice diplomových prací, [4] a [5], se dále zaměřuje na identifikaci a tvorbu matematického modelu stávající konstrukce a následným návrhem stavového regulátoru. I tyto poznatky budou později využity a implementovány do nové verze robotu.

Obrázek 1: Konečná podoba první verze quadrocopteru

V současné době se pozornost přesměrovala k tvorbě zcela nového řešení po konstrukční, hardwarové i programové části. Došlo k tomu z důvodu nedostatků v první verzi, které bránily dalšímu rozvoji projektu. Řeč je zejména o novém hardwaru, který nabízí vyšší výpočetní výkon díky výkonnějšímu mikrokontroléru. Právě ten byl limitující, což potvrzuje závěr práce [4]. Přínosem, především pro operace ve vnějším

prostředí, bude také zcela nová uhlíková konstrukce vybavená výkonnějšími motory, což umožní operace s přídavnými zařízeními. Nová verze robotu je na ústavu označena jménem Uranus (Obrázek 2).

Tato diplomová práce si klade za úkol vytvoření nového řídicího systému pro popsaný létající prostředek. Jeho základem je výběr vhodného řídicího mikrokontroléru, který poskytne dostatečný výpočetní výkon, hardwarové a komunikační prostředky, dostatek programové a operační paměti a v neposlední řadě zázemí v ohledu vývojových prostředí, dokumentací, knihoven a technické podpory. Vhodná volba je zde důležitá díky budoucímu vývoji, jelikož přechod na novou platformu s sebou přináší mnoho komplikací a nelze ji provádět v každé nové verzi. Zvoleným mikrokontrolérem tak je 32bitový LM3S8962 s jádrem ARM Cortex™-M3 od firmy Texas Instruments.

Obrázek 2: Nová verze robotu označená jménem Uranus

Další velkou kapitolu tvoří programové řešení systému na zvoleném mikrokontroléru. Práce se zabývá možnostmi programového řízení embedded aplikací se zaměřením na možnosti operačních systémů. Jeden z vhodných kandidátů, FreeRTOS, je popsán podrobněji a následně také implementován a použit pro řízení.

Změnou od první verze robotu prošlo také dálkové řízení. To nyní využívá digitálního přenosu, který zajišťují Zig-Bee moduly pracující v pásmu 2,4 GHz. Díky tomu je možné kromě řídicích povelů přenášet jakékoliv data v obou směrech. S tím úzce souvisí začlenění do nadřazeného řídicího systému Cassandra-WPF, který je na ústavu vyvíjen. Dálkové řízení tak probíhá za použití PC s tímto systémem a připojeným HID ovládacím zařízením. Komunikaci a systému Cassandra-WPF je věnována zvláštní kapitola.

V neposlední řadě je použita nová inerciální jednotka VectorNav VN-100T, která je vybavena pokročilým zpracováním dat ze snímačů pomocí Kalmanova filtru a poskytuje přímo potřebné údaje o orientaci robotu v prostoru. Jedinými periferiemi, které nebyly doposud nahrazeny vhodnějšími, jsou regulátory použitých BLDC motorů.

U současných je problém s proudovým omezením, díky kterému nebude možné využít plného výkonu motorů a tím plného zatížení užitečným nákladem.

Poslední část je věnována identifikaci základních parametrů nové konstrukce robotu a vytvoření jednoduchého matematického modelu. Pro daný model jsou dále navrženy regulátory a implementovány do řídicího systému jako demonstrace jeho funkčnosti.

Důležité je zmínit, že tato práce vznikala po boku diplomové práce [6], která se zabývá systémem automatického přistávání. Její jedna část, využívající neinerciálních snímačů, bude později implementována do řídicího systému popsaného v této práci. Druhá část, využívající kamerového systému, bude implementována do zvláštního průmyslového PC. Podrobněji jsou v ní také popsány hardwarové prostředky, kterými se tato práce zabývá pouze okrajově.

2 PROGRAMOVÉ ŘEŠENÍ

Tato kapitola se zaměřuje na způsoby, jakými lze řešit řídicí program pro mikrokontrolér, nebo obecně pro jednoúčelové (embedded) aplikace. Některé aplikace si mohou vystačit s velmi jednoduchým softwarovým řešením a využívat pro chod programu především dostupné hardwarové prostředky, ale u složitějších aplikací, s mnoha úlohami, se takové řešení může stát velmi nepřehledným. Pro takové aplikace je vhodné použít složitější softwarové řízení chodu programu, které umožní mít kontrolu nad jednotlivými úkoly a plnit časové a prioritní požadavky. Vrcholem takového řešení jsou operační systémy, které poskytují velmi univerzální základ a prostředky pro tvorbu samotné aplikace.

2.1 Supersmyčka

Řízení programu v supersmyčce, neboli nekonečné smyčce, je většinou vhodné pouze pro jednoduché aplikace, které si vystačí se sekvenčním vykonáváním jednotlivých úloh, které zároveň nejsou časově kritické. Hodí se zejména pro řízení aplikací s fixně danou posloupností vykonávání.

Základem takového řízení je nekonečná smyčka. V této smyčce jsou úlohy, realizované funkcemi a obsluhující určitý logický celek, řazeny za sebou a vykonávány postupně, nebo jsou volány při splnění určitých podmínek. První způsob je jednodušší variantou na realizaci, nenabízí ovšem žádné možnosti větvení. Ukázkou je následující kód v jazyce C:

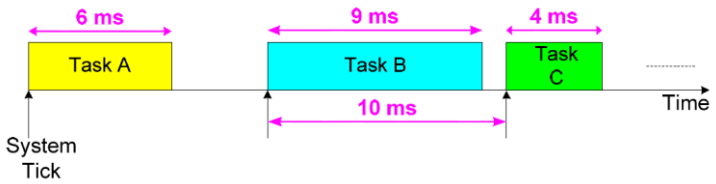
```
void main(void)
{
    Init();
    while(1)
    {
      Task_A();
      Task_B();
      Task_C();
    }
}
```

Funkce `Init()` je volána pouze jednou po startu programu a slouží pro inicializaci periferií a funkce `Task_A` – `Task_C` jsou volány cyklicky v nekonečné while smyčce v pevně daném pořadí. Doba trvání jednoho cyklu je dána součtem časů, který potřebují jednotlivé úlohy. Z toho vyplývá hlavní nevýhoda tohoto řešení, kterou je nemožnost přesného časování úloh, zejména pak, když mají úlohy proměnnou dobu vykonávání. Přidáním další úlohy do smyčky se perioda vykonávání opět sníží. Časově kritické úlohy by proto měly být obsluhovány formou přerušení od časovačů, případně od externího přerušení. Těmito prostředky jsou vybaveny všechny dnešní mikrokontroléry.

I tento jednoduchý přístup k řízení může být dále vylepšován, zabývá se jím např. práce [7]. Cílem je zajistit neměnnou periodu smyčky (periodu vykonávání jednotlivých úloh) i za předpokladu proměnné doby potřebné pro úlohu. To může být zajištěno časovačem, který dovolí vykonávat následující úlohu až po určité, pevně stanovené době. Tato doba musí být větší než maximální doba potřebná pro vykonání časově nejnáročnější úlohy. Příkladem řešení je následující kód:

```
void main(void)
{
    Init();
    while(1)
    {
       DelayStart();
       Task_A();
       DelayWait(10);
       Task_B();
       DelayWait(20);
       Task_C();
       DelayWait(30);
    }
}
```

Každá volaná funkce (úloha) má k dispozici stejné časové okno trvající 10 ms, čímž je v tomto případě zajištěna perioda vykonávání přibližně 30 ms. Tímto způsobem je zajištěno předvídatelné a neměnné časování. Situaci znázorňuje Obrázek 3.



Obrázek 3: Časování sekvence úloh v supersmyčce pomocí časového okna [7]

Omezením popsaného řešení supersmyčky je fakt, že pořadí úloh je fixní a nemůže dojít k žádnému větvení. To řeší přístup, kdy jsou úlohy volány až při splnění určitých podmínek. Každá úloha se tak skládá ze samotné akce a podmínky pro přechod do jiné úlohy. Program může být realizován pomocí příkazu switch:

```
void main(void)
{
    Init();
    int state = 0;
    while(1)
    {
       switch(state)
       {
           case 0: Task_A(); Conditions_A(); break;
           case 1: Task_B(); Conditions_B(); break;
           case 2: Task_C(); Conditions_C(); break;
```

```
            default: TaskDef(); ConditionsDef(); break;
        }
    }
}
```

Funkce `Conditions_x` musí obsahovat podmínku pro přechod a cílový stav. Může vypadat následovně:

```
void Conditions_A(void)
{
    if(temp_sensor) state = 1;
    if(pressure_sensor) state = 2;
}
```

Tímto způsobem se z původně pevné sekvence úloh stal stavový automat běžící v nekonečné smyčce. Stále zde ovšem platí problémy s časováním úloh, jak bylo uvedeno výše.

Nelze obecně říci, že programy řešené supersmyčkou nejsou vhodné pro výkonnostně náročné aplikace. Naopak, nejsou zde žádné, nebo pouze minimální režie pro správu běhu, veškerý výkon tak může být efektivně využit. Hlavní výhodou řešení je jednoduchost a průhlednost realizace a rychlost běhu, nevýhodou je především složitost časování a nemožnost zaručení real-time odezvy. Z toho důvodu jsou časově kritické události téměř vždy obsluhovány formou přerušení, ať už z vnějšího, či vnitřního zdroje. Počet hardwarově implementovaných zdrojů přerušení je ovšem vždy omezen, proto bývá u komplexnějších aplikací nezbytné zavést určitou formu softwarového řízení.

2.2 Přerušením řízené aplikace

Představme si aplikaci, kde je potřeba obsluhovat přerušení z vnějších zdrojů a zároveň množství časově řízených událostí s různou periodou. Například čtení hodnoty výstupu analogového snímače teploty každých 100 ms, aktualizovat střídu PWM signálu každých 20 ms, odesílání dat na sériovou linku každých 500 ms atd. Za použití pouze jednoho hardwarového časovače lez vytvořit jednoduchá správa časových událostí. Systémový časovač *SysTick* je nastaven tak, aby vyvolával přerušení každou 1 ms. Přerušovací rutina obsahuje tabulku událostí, které se mají vykonat a tabulku period, s kterými se mají vykonávat. Při každém vyvolání přerušení jsou všechny periody v tabulce sníženy o jedničku a jakmile jedna z nich dosáhne nuly, je vykonána příslušná událost. Následující kód je příkladem realizace přerušení v jazyce C pro LM3S8962:

```
void SysTickIntHandler(void)
{
    char i;

    /* Pole period jednotlivych udalosti. */
    const static unsigned long ulTimingTable[3] = {
```

```
        100,
        20,
        500
    };

    /* Pole slouzi k urceni pocatecniho intervalu jednotlivych
    udalosti. Dale je plneno hodnotami z ulTimingTable. */
    static unsigned long ulTimingTableTemp[3] = {
        1000,
        1000,
        1000
    };

    /* Pole ukazatelu na funkce pro jednotlive udalosti. */
    static void (*eventsTable[])() = {
        TempEvent,
        PWMEvent,
        UARTEvent
    };

    /*Pri kazdem vyvolani SysTickIntHandler se dekrementuji vsechny
    polozky v ulTimingTableTemp[]. Jakmile je nejaka rovna 0, zavola
    se funkce z eventsTable[] odpovidajiciho indexu. Do pole
    ulTimingTableTemp[] se nahraje hodnota periody z
    ulTimingTable[].*/
    for (i=0; i < sizeof(ulTimingTableTemp)/4; i++)
    {
        ulTimingTableTemp[i]--;
        if (ulTimingTableTemp[i]==0)
        {
            eventsTable[i]();
            ulTimingTableTemp[i]=ulTimingTable[i];
        }
    }
}
```

Pole `ulTimingTable` zde slouží pro trvalé uložení period událostí, v poli `ulTimingTableTemp` jsou uloženy aktuální hodnoty, které se dekrementují a zároveň slouží k určení počáteční periody. V poli `eventsTable` jsou uloženy ukazatele na funkce, které představují události. Jejich pořadí zároveň odpovídá pořadí vykonání v případě, kdy se má vykonat více událostí v jedné obsluze přerušení. Po vykonání příslušné události je do pole `ulTimingTableTemp` uložena hodnota periody.

Představené řešení má několik úskalí, které je třeba mít na paměti. Pro správný chod programu je nutné, aby obsluhy všech událostí dohromady byly kratší než 1 ms, což je zde perioda systémového časovače. Může totiž dojít k situaci, kdy v jednom přerušení dosáhnou všechny dekrementované periody nuly. I toto úskalí by ovšem šlo dále softwarově vhodně ošetřit. Faktem ale zůstává, že všechny implementované časové události budou vždy probíhat na stejné hardwarové prioritě.

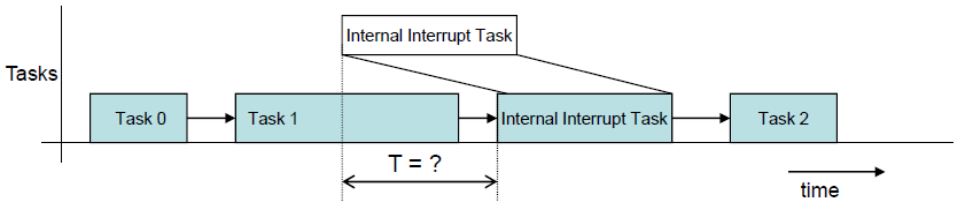
2.3 Operační systémy

Operační systémy tvoří složité softwarové řešení pro řízení běhu programu. Jeho hlavním úkolem je správa systémových prostředků, jimiž jsou procesor, paměti a periferie. Zajišťuje, aby byly co nejefektivněji vykonány všechny požadované úkoly s ohledem na jejich prioritu. Pokud je prioritou OS reagovat na události a provádět výpočty v definovaných časových intervalech, bavíme se o operačních systémech reálného času – RTOS [10].

2.3.1 Základní principy

Složitější aplikace často vyžadují zpracování více úloh najednou. Toho s použitím jednoho procesoru nejde dosáhnout, jelikož v jeden časový okamžik může být zpracovávána pouze jedna úloha. Paralelismu zpracování je dosaženo rychlým střídáním běžících úloh, což vytváří zdání současného běhu. Tato schopnost se obecně nazývá multitasking. Podle způsobu přidělování času úlohám se operační systémy (resp. multitasking) dělí na dvě základní kategorie [8].

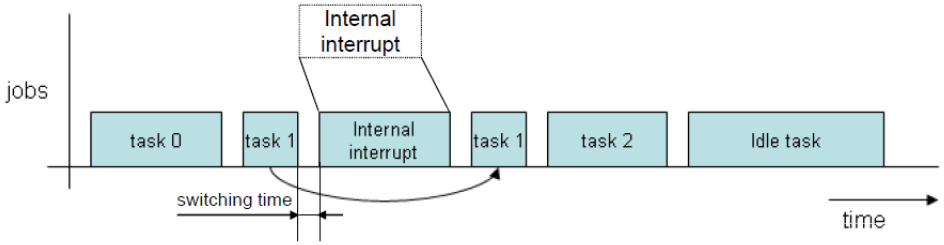
Nepreemptivní OS – jednotlivé úlohy dostávají čas ke svému vykonání postupně s ohledem na jejich pořadí, nebo prioritu. Úloha, která se začne zpracovávat, skončí až v okamžiku, kdy je dokončena, nebo se sama neukončí. OS nemá možnost běh úlohy přerušit, či ukončit z důvodu potřeby vykonání jiné úlohy, byť má vyšší prioritu. Hlavním znakem takových OS je tedy fakt, že nedochází k přerušování úloh – nepreemptivní (kooperativní) multitasking. Obrázek 4 znázorňuje situaci, kdy v průběhu vykonávání Úlohy 1 vznikl požadavek na přerušení a vykonání přerušovací rutiny. Jelikož nemůže dojít k preempci, je vykonána až po ukončení Úlohy 1.



Obrázek 4: Časová posloupnost úloh v nepreemptivním OS [8]

Čas T mezi vznikem požadavku na přerušení a začátku vykonávání požadované úlohy je nepředvídatelný a chování systému je časově nedeterministické. To je důvod, proč u tohoto druhu multitaskingu nemůže být dosaženo real-time odezvy. Výhodou nepreemptivních OS je jednoduchá implementace s malými systémovými nároky. Samotné přepínání úloh by totiž znamenalo ukládání jejich aktuálního stavu do zásobníku a zvyšovaly by se tak celkové režie. Výhodou oproti sekvenční supersmyčce je více úlohové prostředí s možností nastavení priorit jednotlivým úlohám.

Preemptivní OS – doplňují popsaný princip o možnost přerušení vykonávané úlohy úlohou s vyšší prioritou. O přidělování času úlohám plně rozhoduje OS. Přijde-li požadavek na přerušení a OS vyhodnotí, že má vyšší prioritu než právě zpracovávaná úloha, úloha je přerušena a je vykonána přerušovací rutina. Čas mezi vznikem požadavku a začátkem vykonávání požadované úlohy se nazývá *switching time* (Obrázek 5). Tento čas je již předvídatelný a je proto možné zajistit real-time odezvu.



Obrázek 5: Časová posloupnost úloh v preemptivním OS [8]

Tato vlastnost ovšem neznamená, že se jedná o RTOS. RTOS musí zajistit, aby reakce systému na příchozí události byly v definovaných (požadovaných) časových intervalech, tzv. *deadlines*. Toto chování nelze zajistit řešením pomocí supersmyčky a nepreemptivního multitaskingu, jelikož v obou případech nemá systém plnou kontrolu nad časováním úloh. Preemptivní OS má na druhou stranu mnohem vyšší systémové nároky, jelikož stav každé úlohy při přerušení musí být uložen a řízení běhu se tak stává složitější a méně průhledné.

Operační systém je tedy složitý program starající se o přidělování systémových zdrojů. Dbá na to, aby byly všechny úlohy řádně dokončeny, dodržuje priority úloh a jejich časové požadavky. V souvislosti s OS jsou rozlišovány základní pojmy.

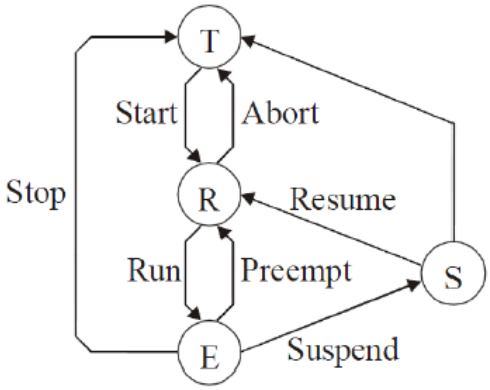
Proces (process) – proces nevykonává žádný exekuční kód, ale je to prostředí obsahující jednotlivé úlohy vykonávané ve vláknech. Proces musí obsahovat tedy alespoň jedno vlákno, aby mohl plnit nějakou úlohu. Každý proces má přidělený svůj paměťový prostor, který je dělen mezi jednotlivá vlákna.

Úloha (task) – tvoří logickou sekvenci operací, které jsou vykonávány nezávisle na jiných úlohách. Úloha může být vykonávána v jednom vlákně, nebo je rozdělena mezi více vláken.

Vlákno (thread) – jednoduchý exekuční blok složený z instrukcí, který vykonává požadovanou činnost pro nadřazenou úlohu. Vlákno pracuje s částí paměti přidělené určitému procesu, který je nadřazený úlohám.

Již výše byla řeč o multitaskingu, neboli o zdánlivém zpracování více úloh paralelně. Je-li k dispozici pouze jeden procesor, v jeden časový okamžik může být zpracovávána pouze jedna úloha. Úloha má přidělené určité časové kvantum pro svůj běh, přičemž ostatní úlohy jsou neaktivní. Úlohy se tak mohou nacházet v různých stavech [9]. Na Obrázku 6 je zobrazen stavový diagram úlohy.

Stavy T (ukončen), R (připraven) a S (pozastaven) reflektují stav mezi úlohou a okolím a stupeň připravenosti pro stav E (běžící). Úloha ve stavu T nemůže být provedena z důvodu čekání na splnění nějaké podmínky. Úloha ve stavu R má již splněny všechny podmínky pro běh, ale čeká na přidělení systémových zdrojů. Ve stavu E se nachází pouze jedna aktivní úloha, která je buď ukončena a přechází do stavu T, nebo přerušena a vrací se do stavu R, či pozastavena z důvodu nesplnění nějaké podmínky. Z pozastaveného stavu S může být opět připravena nebo ukončena.



Obrázek 6: Stavový diagram úlohy [9]

Stavový diagram také ukazuje složitost řízení běhu programu. Musí být počítáno s tím, že samotné řízení potřebuje určitý strojový čas procesoru a se zvyšujícími se nároky na paměť, do které jsou neustále zapisovány a vyčítány stavy jednotlivých úloh.

Část OS, která provádí všechny operace správy úloh, se nazývá systémové jádro – *kernel*. V případě RTOS se musí postarat také o striktní dodržování stanovených *deadlines* úloh, aby byla zaručena real-time odezva.

Operační systémy, jaké známe z PC, jsou mnohem složitějšími programy umožňující uživateli vykonávat široké spektrum činností pomocí instalovaných programů a využívat nejrůznější periferie. Pro jednoúčelové aplikace se však jedná o složité a drahé řešení. Existují ovšem i mnohem jednodušší varianty OS zaměřené na embedded aplikace, které poskytují určitý komfort známý z OS pro PC i při programování mikrokontrolérů.

2.3.2 OS pro mikrokontroléry

Běžné mikrokontroléry Harvardské architektury disponují oproti PC velmi omezeným výpočetním výkonem procesoru, který je u embedded aplikací taktován v řádech jednotek až desítek MHz a malou kapacitou pamětí, které se pohybují v jednotkách až stovkách kB. Z toho vyplývá, že operační systémy pro ně určené musí mít mnohem menší systémové nároky a zpravidla jsou nesrovnatelně jednodušší. To vychází také

z faktu, že nejsou určeny pro obecné použití jako na PC, ale představují způsob řešení jednoúčelové aplikace.

I přes jejich velké rozdíly se stále jedná o systémy, které mají za úkol spravovat a přidělovat systémové prostředky a poskytovat funkce pro tvorbu aplikace. Členění na procesy, úlohy a vlákna, jak bylo popsáno výše, je zpravidla zjednodušeno na pouhé exekuční bloky nazývané *tasky* - úlohy. Kromě funkcí pro spravování jednotlivých úloh systémy nabízejí funkce pro jejich synchronizaci, komunikaci, funkce pro časování a práci se systémovými prostředky. Poskytují tak prostředí, díky kterému je výsledná aplikace přehlednější, přenositelnější a v neposlední řadě je její tvorba příjemnější.

Operačních systémů pro mikrokontrolér LM3S8962, který je použit v této práci (podrobněji popsán v kapitole Mikrokontrolér LM3S8962), je k dispozici hned několik. Výběr byl však zúžen na pouhé dva, FreeRTOS a CoOS. FreeRTOS [11] je velmi rozšířený především díky široké podpoře mikrokontrolérů a vývojových prostředí. Podporuje 31 architektur, kde nabízí v 6 až 10 kB zdrojových kódů základní prostředky pro pohodlnou tvorbu aplikace. OS je přímo doporučován výrobcem použitého mikrokontroléru Texas Instruments a jeho nespornou výhodou je podrobná dokumentace, množství ukázkových zdrojových kódů a široká komunita. Právě z těchto důvodů je v práci použit a je mu věnována kapitola Operační systém FreeRTOS. Operační systém CoOS tvoří plnohodnotnou alternativu, jelikož je zaměřen právě na mikrokontroléry s jádrem ARM Cortex™-M. Jeho jádro si vystačí s necelým 1 kB ROM paměti, přičemž nabízí obdobné nástroje pro správu úloh a jejich synchronizaci a komunikaci jako systém FreeRTOS. Více informací a popis API funkcí poskytuje výrobce online [12].

3 OPERAČNÍ SYSTÉM FREERTOS

Tato kapitola je zaměřena na seznámení s operačním systémem pro embedded aplikace FreeRTOS, na popis jeho charakteristik, API funkcí a způsobu implementace.

3.1 Základní charakteristiky

FreeRTOS je multiplatformní operační systém reálného času určený pro embedded aplikace, který podporuje 31 architektur mikrokontrolérů výrobců jako je Atmel, Freescale, Microchip, ST Microelectronic, Texas Instruments a další [11]. Je vyvíjen společností Real Time Engineers Ltd. od roku 2003 a jeho cílem je poskytnout systém průhledný a jednoduchý k použití, s množstvím ukázkových aplikací a to vše pod svobodnou licencí. Zdrojové kódy jsou psány v jazyce C a jsou logicky děleny na funkční celky tak, aby byly nezávislé na architektuře a umožňovaly snadný přechod na nové verze. Dle konfigurace si systém vystačí s 6 až 10 kB paměti. Systém nabízí preemptivní plánovač (*scheduler*) s podporou úloh (*task*) a co-rutin, prostředky pro synchronizaci a komunikaci mezi úlohami (mutexy, semafory, fronty), nástroje pro ladění systému a správu paměti (detekce přetečení zásobníku apod.). FreeRTOS je dostupný zdarma pod licencí open-source a je ho možné používat i v komerčních aplikacích. Kromě něj jsou nabízeny další dvě verze systému:

OpenRTOS – komerční verze systému, která kromě technické podpory zahrnuje prostředky pro práci s USB či TCP/IP.

SafeRTOS – verze určená pro *safety-critical* aplikace splňující úroveň SIL 3 standardu EIC 61508.

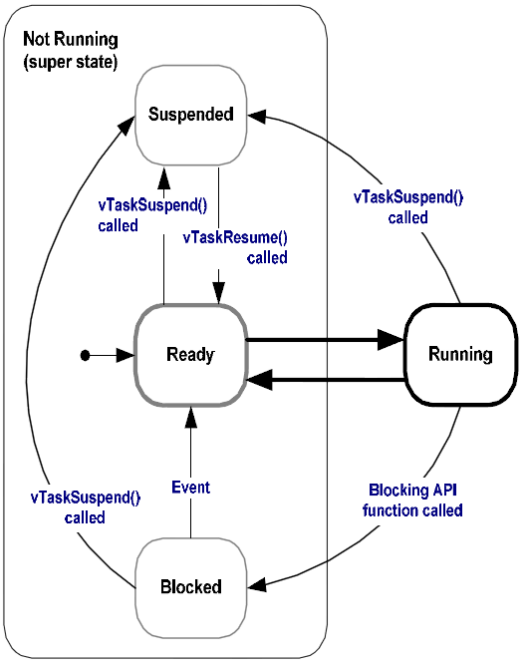
3.2 Jádro systému

Oproti operačním systémům pro PC je FreeRTOS tvořen pouze jádrem systému obsahující plánovač úloh a soubor API funkcí pro práci s nimi. Řízení běhu programu je zde zjednodušeno natolik, že nepracuje s procesy, jakožto celky skládajících se z úloh a vláken, ale pouze se samotnými úlohami, které tvoří exekuční bloky. Procesy v PC mají přidělený vlastní ohraničený paměťový prostor, kterého využívají jeho úkoly a vlákna. I zde mají úlohy přidělen svůj paměťový prostor, ale celkově se všechny úlohy dělí o operační paměť mikrokontroléru a jsou limitovány její velikostí. Velikost paměti, kterou úloha potřebuje, se definuje při její vytváření.

3.2.1 Plánovač

Plánovač je důležitou součástí jádra každého více úlohového OS. Jeho úkolem je vybírat úlohu, které bude v daný okamžik přidělen procesor, neboli která se stane

aktivní. FreeRTOS využívá prioritní plánování, kdy každá úloha má přidělenou prioritu. Stejně jako v předchozí kapitole i zde je definován stavový diagram úlohy, který je zobrazen na Obrázku 7. Principiálně obsahuje stejné stavy, trojici nečinných a jeden aktivní, ve kterém je právě vykonávaná úloha. Úlohy ve stavu pozastaven (*suspended*) nejsou k dispozici plánovači, do stavu mohou být vloženy nebo vyjmuty uvedenými funkcemi. Úloha ve stavu připraven (*ready*) čeká na přidělení plánovačem, aby se stala aktivní. Blokované úlohy (*blocked*) čekají na časovou (časová blokace), nebo synchronizační událost (mutex, semafor, fronta zpráv), aby mohly přejít do stavu připraven [13].



Obrázek 7: Stavový diagram úlohy ve FreeRTOS [13]

3.2.2 Úlohy

Jak již bylo zmíněno, výsledná aplikace se skládá z úloh, což jsou jednotlivé exekuční bloky kódu, tvořící logické celky. Programově je v jazyce C každá úloha tvořena samostatnou funkcí, která má inicializační část a část tvořenou nekonečnou smyčkou. Smyčka obsahuje zdrojový kód k vykonání a za použití systémových API funkcí je definováno její chování v systému. Ukázkou je následující kód:

```
void vLEDTask( void *pvParameters)
{
    /* Inicializacni cast – vytvoreni promennych, inicializace
    periferii. */
    long lPinStatus;
```

```
    for(;;)
    {
          /* Nekonecna smycka obsahuje funkcni kod ulohy, v tomto
          pripade nacteni stavu pinu, zmenu stavu a opetovny zapis.
          */
          lPinStatus = GPIOPinRead(GPIO_PORTF_BASE, GPIO_PIN_0);
          lPinStatus = ~lPinStatus;
          GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_0, lPinStatus);

          /* API funkce vTaskDelay zpusobi, ze bude uloha pred dalsim
          vykonanim 500 ms ve stavu blocked. */
          vTaskDelay( 500 / portTICK_RATE_MS);
    }
}
```

Ukázkový kód je obyčejná funkce, o které plánovač prozatím neví. Aby byla funkce skutečně v systému vytvořena, musí být registrována pomocí následující API funkce:

```
portBASE_TYPE xTaskCreate( pdTASK_CODE pvTaskCode,
                           const signed char * const pcName,
                           unsigned short usStackDepth,
                           void *pvParameters,
                           unsigned portBASE_TYPE uxPriority,
                           xTaskHandle *pxCreatedTask
                           );
```

První parametr je právě C funkce, která odpovídá nově vytvořené úloze (v našem případe `vLEDTask`). Druhý parametr je string obsahující název úlohy pro účely ladění. Třetí parametr je velikost paměťového prostoru v paměti RAM, který bude mít funkce k dispozici. Její rozměr je udáván ve slovech (u LM3S8962 má slovo velikost 4 B). Čtvrtý parametr je parametr předaný funkci při jejím vytvoření. Pátý parametr funkce definuje její prioritu, kde nejnižší hodnota odpovídá nejnižší prioritě. Poslední parametr je ukazatel, do kterého je uložena právě vytvořená funkce.

Tímto způsobem jsou jednotlivým úlohám přiřazeny odpovídající C funkce. Poslední věcí, která je nutná udělat pro běh aplikace, je aktivace plánovače. To je provedeno funkcí `vTaskStartScheduler`.

Běžně jsou jednotlivé úlohy vytvořeny po inicializaci hardwaru mikrokontroléru a následně je spuštěn plánovač. Úlohy je možné samozřejmě vytvářet i v průběhu běhu aplikace, stejně tak je možné je mazat.

Zvláštní typem úlohy je *IDLE task*. *IDLE task* je vytvořena automaticky při startu plánovače a systém v ní tráví procesorový čas v případě, kdy není třeba vykonávat žádnou jinou úlohu.

3.2.3 Časování

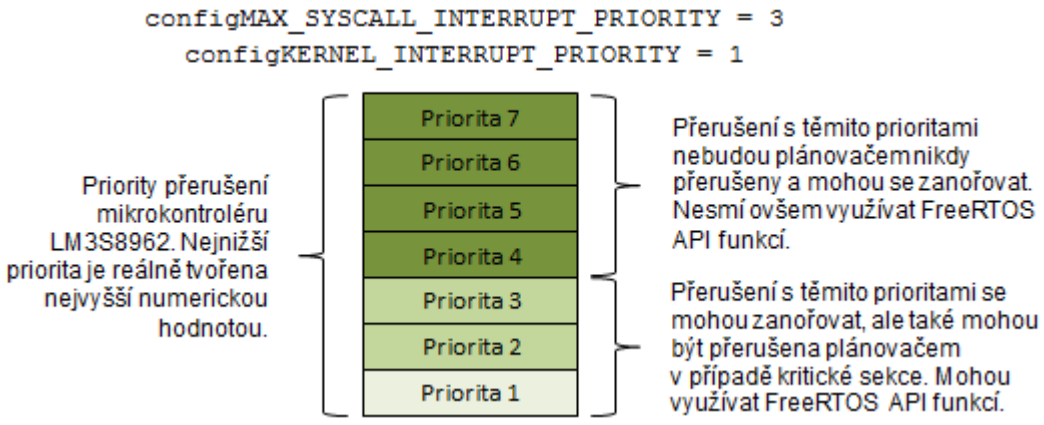
Plánovač operačního systému ke svému běhu nutně potřebuje přesné časování, které je zpravidla zajištěno systémovým časovačem mikrokontroléru (*SysTick*). Perioda časovače představuje základní časovou jednotku pro časování úloh. Časování systému

se nastavuje v konfiguračním souboru `FreeRTOSConfig.h` pomocí parametrů `configCPU_CLOCK_HZ` a `configTICK_RATE_HZ`.

3.2.4 Přerušení a priority

K pochopení principu fungování jádra systému a návrhu aplikace je třeba porozumět mechanismu přerušení. Již bylo zmíněno, že plánovač systému využívá přerušení od systémového časovače. Málokterá aplikace si ovšem vystačí bez použití dalších zdrojů přerušení, například reakce na vnější události. U použitého mikrokontroléru je možné každému přerušení přiřadit číselnou prioritu, stejně tak jsou přiřazovány softwarové priority jednotlivým úlohám. Proto je potřeba jasně definovat, kdy má jaká část aplikace přednost.

Mikrokontrolér LM3S8962 umožňuje jednotlivým vektorům přerušení přiřadit priority 0 – 7, přičemž menší číslo odpovídá vyšší prioritě. Zároveň umožňuje zanořování přerušení. Hardwarová priorita systémového časovače je nastavena v konfiguračním souboru hodnotou `configKERNEL_INTERRUPT_PRIORITY`. Dále je definována mezní priorita přerušení, ve kterých mohou být používány API funkce systému a které mohou být potlačeny, pokud se aplikace nachází v kritické sekci. Odpovídá jí parametr `configMAX_SYSCALL_INTERRUPT_PRIORITY`. Přerušení s vyšší prioritou, než je tato hodnota, již nemohou využívat API funkcí a nemohou být ovlivňovány plánovačem. Situaci znázorňuje Obrázek 8.



Obrázek 8: Grafické znázornění rozdělení hardwarových priorit

Softwarové priority, které jsou nastavovány při vytváření úloh, mají význam pouze pro mechanismus plánovače. Ten běží se zmíněnou hardwarovou prioritou `configKERNEL_INTERRUPT_PRIORITY`.

3.3 Přehled API funkcí

Tato kapitola stručně představuje API funkce systému FreeRTOS, které jsou k dispozici pro tvorbu aplikace. Již bylo představeno, jakým způsobem jsou vytvářeny úlohy a jak je aktivován plánovač. Systém dále poskytuje funkce pro správu úloh, jako pozastavení (*suspend*) a obnovení (*resume*) úlohy, změnu její priority, funkce pro časování úloh a mazání. Dále jsou k dispozici funkce pro komunikaci mezi úlohami, jejich synchronizaci, funkce pro ochranu zdrojů a použití časovačů. Kompletní specifikace API funkcí je dostupná online [11] nebo [13].

3.3.1 Fronty zpráv

Fronty jsou základním nástrojem pro bezpečné předávání dat mezi jednotlivými úlohami (*intertask communication*), případně mezi obsluhami přerušení (*ISR*) a úlohami. Jsou tvořeny FIFO frontou, kdy nová data jsou primárně řazena na její konec, což ale nemusí být podmínkou. Při vytváření fronty tak musí být definován datový typ, který bude předáván, a to z důvodu definování velikosti jednoho pole fronty a dále maximální počet prvků, které se do fronty vlezou.

Vkládání nových dat do fronty probíhá formou kopie, z čehož vyplývají její nároky na operační paměť. Proto je vhodné vytvářet fronty pouze takové velikosti, jaké je potřeba. Při příjmu dat jsou data z fronty odebrána. Tímto způsobem je zajištěno jednoduché předávání dat mezi dvojicí úloh, které navzájem ke svým lokálním proměnným nemají přístup.

Fronty mohou být využity také pro blokování a synchronizaci úloh. Příkladem může být situace, kdy jedna úloha data zpracovává a druhá odesílá výsledek na komunikační kanál. Úloha pro odeslání je nečinná až do doby, kdy se ve frontě objeví zpracovaná data.

Mezi základní API funkce patří `xQueueCreate`, která přebírá parametr o velikosti dat a počtu prvků. Fronty jsou zpravidla vytvářeny před startem plánovače. Funci pro zaslání dat do fronty, `xQueueSend`, musí být předán parametr obsahující konkrétní frontu, ukazatel na data a časový údaj, který definuje, jak dlouho může funkce čekat v případě, kdy je fronta plná. Data z fronty mohou být vyzvednuty pomocí funkce `xQueueReceive`, která stejně tak přebírá parametr o konkrétní frontě, ukazatel na cílový buffer a časový údaj, jak dlouho může funkce čekat na nová data. Nastavení časového parametru na hodnotu `portMAX_DELAY` dochází k blokaci úlohy po dobu, než se objeví nová data.

3.3.2 Semafory

Semafory jsou určeny pro vzájemnou synchronizaci úloh, případně pro synchronizaci obsluh přerušení a úloh. Jsou rozlišovány dva typy semaforů, binární semafor (b*inary semaphore*) a čítající semafor (*counting semaphore*).

Binární semafor se dá představit jako fronta s rozměrem jedna, která může být buď prázdná, nebo plně obsazená. To odpovídá dvojici jeho stavů, kdy semafor není, nebo je nastaven. Stejně jako fronty, i binární semafory mohou být použity pro blokování neaktivních úloh. Čekající úloha může být blokována do doby, než je jinou úlohou semafor nastaven. Binární semafory se často používají pro synchronizaci s přerušením. Představme si situaci, kdy po stisku klávesy mikrokontrolér reaguje vstupem do přerušovací rutiny. Ty mají běžně vyšší prioritu, než je systémová, proto jejich obsluha musí být co nejkratší. V přerušovací rutině je nastaven příslušný semafor a tím je odblokována úloha, která se stará o příslušnou akci.

Před samotným vytvořením semaforu je potřeba vytvořit proměnnou typu `xSemaphoreHandle`. Ta je předána jako jediný parametr funkci `vSemaphoreCreateBinary`, která jej v systému vytvoří. Za použití vytvořeného handleru pak mohou být použity funkce pro nastavení semaforu `xSemaphoreGive` a pro příjem `xSemaphoreTake`. Ta přebírá ještě časový parametr, jak dlouho má funkce na příjem čekat, případně být blokována hodnotou `portMAX_DELAY`.

Čítající semafory mohou nabývat více hodnot než jedna. Toho se využívá například pro čítání událostí. Jedna úloha inkrementuje semafor vždy při příslušné události, druhá úloha postupně semafor dekremetuje při zpracování událostí.

3.3.3 Mutexy

Mutexy jsou speciálním druhem binárních semaforů, které jsou využívány pro ochranu zdrojů a systémových prostředků. Zabraňují například, aby v jeden okamžik přistupovalo k jedné komunikační sběrnici více úloh. Takovým kritickým úsekům kódu musí vždy přecházet žádost o mutex. Vytvořený semafor má k dispozici pouze jeden mutex, který na žádost přiděluje a po dokončení kritické činnosti je mu opět navrácen a je k dispozici jiné úloze. Pokud úloha žádá o mutex, který není k dispozici, je blokována do té doby, než jiná úloha dokončí práci s chráněnými prostředky a mutex vrátí.

Vytváření mutexu je podobné jako v předchozím případě. Je vytvořena proměnná typu `xSemaphoreHandle`, která je použita pro vytvoření mutexu následujícím způsobem:

```
xSemaphore = xSemaphoreCreateMutex();
```

Žádost o mutex probíhá pomocí funkce `xSemaphoreTake`, které po vykonání kritického kódu vždy musí následovat vrácení mutexu pomocí `xSemaphoreGive`.

3.3.4 Časovače

FreeRTOS poskytuje API funkce pro práci se softwarovými časovači, které mohou být obecně použity pro časování jednorázových i periodických událostí. Událost je definována v tzv. *timer's callback function* a je vykonána po vypršení časovače. Doba mezi spuštěním časovače a obslužnou událostí se nazývá perioda časovače.

Jako všechny systémové nástroje musí být i časovače před samotným použitím v systému vytvořeny. Může se tak dít ještě před zapnutím plánovače, nebo i za běhu aplikace. Při vytváření musí být definována především perioda časovače, informace, zda se jedná o jednorázový časovač či periodický a přiřazena obslužná funkce. Dále jsou definovány API funkce pro spuštění a zastavení časovače, pro změnu periody, restart a smazání časovače.

3.4 Zdrojové soubory

Již bylo dříve zmíněno, že FreeRTOS je vytvořen v jazyce C a jeho klíčovou vlastností je podpora množství architektur mikrokontrolérů. Z toho důvodu je jádro systému nezávislé na platformě, což umožňuje jednoduchý přechod na novou verzi a případně také na jinou architekturu a usnadňuje tak distribuci systému. Systém by ovšem nemohl fungovat, kdyby nebyl propojen s hardwarovými prostředky mikrokontroléru. Proto jsou součástí systému, kromě jeho jádra, také zdrojové soubory, které jsou individuální pro každou architekturu a zajišťují zmíněné propojení. Tyto soubory se zároveň liší s ohledem na použitý kompilátor/vývojové prostředí. Všechny zdrojové soubory jsou volně stažitelné z internetových stránek výrobce [11].

Funkční jádro systému se v základu skládá z pouhých tří souborů: `task.c`, `queue.c` a `list.c` (v případě použití co-rutin ještě ze souboru `croutine.c`). Ty jsou umístěny v adresáři FreeRTOS/Source. Dále v podadresáři FreeRTOS/Source/Portable se nachází zdrojové soubory pro danou architekturu a kompilátor. Individuální zdrojový soubor se nazývá `port.c`. Důležitým souborem, o kterém již byla dříve řeč, je konfigurační `FreeRTOSConfig.h`, který obsahuje veškeré nastavení systému.

Podrobněji se implementací systému pro daný mikrokontrolér LM3S8962 zabývá kapitola Vytvoření FreeRTOS aplikace, kde je popsáno jeho nastavení, tvorba aplikace a dále otestována jeho funkčnost a paměťové nároky.

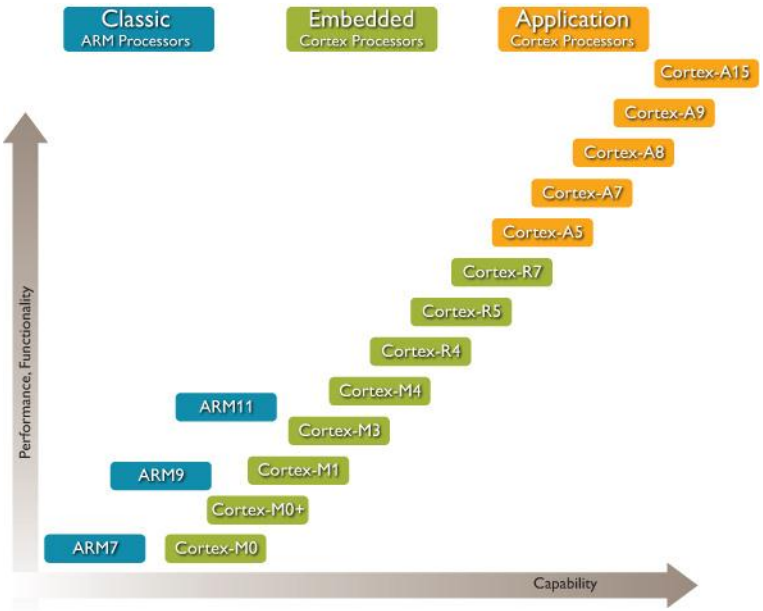
4 HARDWARE

Předešlé kapitoly prozatím popisovaly softwarovému řešení řídicího systému. Byly popsány způsoby řešení programu pro mikrokontroléry, kde byla zvláštní pozornost věnována embedded operačním systémům a především systému FreeRTOS. Návrh softwarového řešení vždy úzce souvisí s použitým hardwarem, proto již bylo dříve několikrát zmíněno, jaký mikrokontrolér bude aplikace využívat. Mikrokontroléru LM3S8962 je věnována zvláštní podkapitola, kde jsou podrobně popsány jeho vlastnosti. Dále se tato kapitola věnuje celkovému hardwarovému řešení, použitým periferiím, a inerciální jednotce VN-100T.

4.1 Mikrokontrolér LM3S8962

S ohledem na požadavky projektu byl vybrán mikrokontrolér s architekturou ARM [14] Cortex™-M3 od výrobce Texas Instruments, typ LM3S8962 [15] z produktové řady Stellaris 8000.

Procesory z rodiny ARM jsou v dnešní době velmi rozšířeny v mnoha odvětvích, od jednoduchých aplikací k průmyslovým a automobilovým, najdeme je v mobilních telefonech, tabletech a noteboocích. Licence na výrobu ARM procesorů má zakoupeno množství předních výrobců procesorů, jako je Samsung, Toshiba, STM, Texas Instruments, Freescale, Fujitsu a další. Procesorů ARM existuje celá řada, rozdělení dle sérií je na Obrázku 9.



Obrázek 9: Přehled procesorů ARM [14]

Classic ARM Processors jsou zatím nejrozšířenější, jelikož jsou určeny pro obecné použití a cenovou dostupností mohou konkurovat jednoduchým 8/16bitovým mikrokontrolérům.

Oblast *ARM Cortex Embedded Processors*, do které zapadají série Cortex-R a Cortex-M, je určena pro aplikace, kde je vyžadován výkon a spolehlivost, pokročilá obsluha přerušení, konektivita, real-time aplikace apod. Najdeme je v inteligentních senzorech, HID zařízeních, automobilových a průmyslových systémech, nebo v bílé technice.

ARM Cortex Application Processors jsou nejvýkonnější ARM procesory pracující na frekvencích až 2 GHz, určeny pro smartphony, tablety, netbooky či digitální televize.

Vybraný mikrokontrolér LM3S8962 pracuje na frekvenci až 50 MHz, obsahuje SRAM a flash paměť s MPU jednotkou pro ochranu paměti, množství vektorů přerušení s nastavitelnou prioritou, víceúčelové vstupně výstupní bloky, konfigurovatelné časovače a watchdog, hardwarovou podporu nejpoužívanějších sériových rozhraní včetně CAN a Ethernetu, AD převodníky a komparátor pro práci s analogovými signály, generátory PWM s QEI enkodéry pro řízení motorů a JTAG rozhraní pro programování a ladění. V následujících kapitolách jsou uvedeny vlastnosti základních periferií obvodu, které jsou znázorněny na blokovém diagramu v Příloze 1.

4.1.1 Paměť

LM3S8962 obsahuje 64 kB SRAM určenou pro data, která podporuje technologii bit-band, umožňující adresaci a přístup k jednotlivým bitům. Dále obsahuje 256 kB flash paměti. Ta je organizována po 1 kB blocích, které mohou být individuálně mazány. Tyto bloky jsou po dvojicích uspořádány do 2 kB bloků, které mohou být chráněny (*read-only, execute-only*).

Paměťový prostor využívá 32bitového adresování. Flash paměť, která slouží jako paměť programu, leží v rozmezí adres 0x0000.0000 až 0x0003.FFFF. Mapováno je tedy 262143 bytů. Od adresy 0x2000.0000 po 0x2000.FFFF leží SRAM paměť, sloužící jako operační paměť, s mapovanými 65635 byty. Paměťový model je podrobněji znázorněn v Tabulce 1.

4.1.2 Přerušení

O správu přerušení se stará jednotka NVIC. Nabízí obsluhu 36 druhů maskovatelných přerušení a nastavení priority 0 až 7 pro každé z nich (prioritu 0 má přerušení s nejvyšší prioritou, systémové přerušení, mají zápornou prioritu). Umožňuje také vytváření skupin priorit a v nich nastavování subpriorit. Kromě resetu, který má nejvyšší prioritu (-3) je zde ještě nemaskovatelné přerušení NMI (-2) od NMI signálu nebo programově spouštěné a Hard Fault přerušení při systémové chybě (-1).

LM3S8962 podporuje zanořování přerušení (*preemption*), které nastává v okamžiku, kdy přijde požadavek na vykonání přerušení s vyšší prioritou, než má právě vykonávané přerušení. Podporuje také *Tail-Chaining*, což je mechanismus, který umožňuje plynulý přechod do čekajícího přerušení po ukončení přerušovací rutiny s vyšší prioritou. Nedochází k zásobníkovým operacím a přechod je tak proveden v pouhých 6ti cyklech (při běžné obsluze v 12ti).

Začátek	**Konec**	**Popis**
Paměti		
0x0000.0000	0x0003.FFFF	Flash paměť
0x2000.0000	0x2000.FFFF	SRAM paměť
0x2200.0000	0x221F.FFFF	Bitové aliasy SRAM paměti
FiRM Periferie		
0x4000.0000	0x4001.FFFF	Watchdog, GPIO Port A-D, SSI0, UART0-1
Periferie		
0x4002.0000	0xDFFF.FFFF	I2C0, GPIO Port E-G, PWM, QEI0-1, Timer 0-3, ADC0 atd.
Private Peripherial Bus (PPB)		
0xE000.0000	0xFFFF.FFFF	SysTick, NVIC, MPU atd.

Tabulka 1: Paměťový model LM3S8962

4.1.3 Systémové periferie

Mezi systémové periferie patří vstupně výstupní GPIO piny, programovatelné časovače a watchdog.

GPIO představují víceúčelové konfigurovatelné vstupně výstupní piny. LM3S8962 je vybaven sedmi fyzickými GPIO bloky, podporující 5 až 42 vstupně výstupních pinů, v závislosti na použitých periferiích. Moduly GPIO umožňují u všech pinů generování přerušení na základě hran a úrovní a také bitové maskování pro operace čtení a zápisu.

Časovače mohou být použity pro čítání nebo časování externích či vnitřních událostí. LM3S8962 obsahuje čtyři moduly víceúčelových časovačů GPTM, přičemž každý z nich poskytuje dva 16 bitové čítače/časovače, nebo jeden 32 bitový časovač. 32 bitový režim může být použit pro hodiny reálného času RTC. 16 bitový režim může být konfigurován jako periodický, nebo použit pro generování PWM signálu.

Watchdog časovač slouží ke generování přerušení nebo resetu při dosažení nastaveného časového intervalu. Slouží k odhalení chyb v programu, nebo selhání externího zařízení. Watchdog obsahuje 32 bitový odpočítávací čítač, programovatelný registr pro načtení hodnoty a registr ochrany. Registr ochrany zabraňuje neúmyslné

změně konfigurace čítače, pokud již byl aktivován. Čítač může být konfigurován tak, aby po prvním dosažení nuly vyvolal přerušení až při druhém resetu.

4.1.4 Komunikační rozhraní

LM3S8962 obsahuje několik synchronních a asynchronních sériových komunikačních rozhraní: dvakrát UART, SSI, I^2C, CAN a Ethernet.

UART rozhraní vychází ze standardního asynchronního RS-232C rozhraní. Obsahuje hardwarový vysílač (paralelní-sériový převodník) a přijímač (sériový-paralelní převodník). Vysílač i přijímač je vybaven odděleným FIFO zásobníkem o velikosti 16x8 bitů. Podporuje maskovatelné přerušení a maximální přenosovou rychlosti 3,125 Mb/s. Oba UART kanály podporují IrDA.

SSI je sériové synchronní rozhraní využívající 4 vodiče, podporující také známé SPI. Umožňuje 4 až 16 bitovou komunikaci, obsahuje hardwarový vysílač a přijímač s nezávislým 8x16 bitovým FIFO zásobníkem. Podporuje režimy master, slave i multi-slave. Frekvence výstupního hodinového signálu je nastavitelná integrovanou děličkou vnitřních hodin.

I^2C je dvouvodičové synchronní sériové rozhraní umožňující obousměrnou komunikaci. Podporuje čtyři módy: master vysílač, master přijímač, slave vysílač, slave přijímač. Nastavitelné jsou dvě přenosové rychlosti: standardní 100 kb/s a rychlá 400 kb/s. Umožňuje generování přerušení od vysílání, příjmu a chyb.

CAN je sériové rozhraní určené pro komunikaci senzorů a řídicích jednotek. Původně bylo navržené pro automobilový průmysl, dnes je díky odolnosti vůči elektromagnetickému rušení používáno v řadě průmyslových aplikací. Maximální rychlost je 1 Mb/s na vzdálenost 40 m. Ta klesá se vzdáleností, např. pro 500 m dosahuje již jen 125 kb/s. U CAN rozhraní je zpráva odesílána vysílačem všem přijímačům, ty se rozhodují na základě identifikátoru. Zpráva může obsahovat 0 až 8 bytů.

Ethernet je rozhraní založené na rámcích umožňující budování počítačových sítí LAN. Je popsáno standardem IEEE 802.3. LM3S8962 má plně integrovanou MAC a fyzickou vrstvu. Podporuje full-duplex a half-duplex, 100 Mb/s a 10 Mb/s módy. Umožňuje nastavení MAC adresy a uživatelských přerušení.

4.1.5 Analogové periferie

Pro práci s analogovými signály obsahuje LM3S8962 A/D převodník a analogový komparátor.

AD převodník poskytuje 10 bitové rozlišení a čtyři vstupní kanály s FIFO zásobníky, které mohou obsluhovat až 8 vstupních signálů se vzorkovací frekvencí až 500 kHz. Každý kanál má nastavitelnou spouštěcí událost a generování přerušení. Umožňuje

hardwarové průměrování až z 64 hodnot, má integrovanou 3 V napěťovou referenci a oddělenou analogovou a číslicovou zem.

Analogový komparátor slouží k porovnávání dvou analogových signálů a jejich logickému vyhodnocení. Využívá buď externího referenčního napětí, nebo interní reference. Jeho výstup může sloužit k nastavení obecného výstupu, nebo generování přerušení pro události AD převodníku.

4.1.6 Periferie řízení pohybu

Pro řízení motorů nabízí LM3S8962 generátory pulsně šířkové modulace PWM a obsluhu optických enkodérů QEI.

PWM nabízí generování obdélníkových signálů s vysokým rozlišením jejich střídy pro dosažení ekvivalentní analogové úrovně signálu. Toho se využívá nejčastěji pro řízení otáček DC motorů, nebo svitu LED diod. LM3S8962 obsahuje tři nezávislé PWM bloky s 16 bitovým up/down čítačem, dvěma komparátory, PWM generátorem a řízením přerušení a spouštění. Dále obsahuje jeden řídící blok, kde je možné konfigurovat polaritu signálu a aktivní výstupní piny.

QEI, známý také jako dvoukanálový inkrementální enkodér, převádí pohyb na signál obsahující pulsy. Na základě počtu pulsů a fáze dvou signálů je možné určit polohu, směr rotace a rychlost. Třetí kanál může být použít pro detekci nulové pozice. LM3S8962 nabízí dva nezávislé QEI moduly, které mohou být použity pro řízení a snímání dvou motorů v jednom časovém okamžiku.

4.1.7 Podpora obvodu

Jednou z výhod použití zvoleného obvodu je jistě jeho softwarová podpora. Programátor může volit mezi řadou vývojových prostředí, kompilerů a debuggerů. Mezi podporované nástroje patří Keil, navržený přímo pro ARM procesory, který má vlastní kompiler a podporu JTAG debuggeru. IAR je rovněž komplexní nástroj vybaven vlastním compilerem a gebuggerem zaměřený na široké spektrum mikrokontrolérů. Nástroje CodeSourcery a CodeRed naopak využívají open source GCC compileru a v neposlední řadě nástroj Code Composer Studio navržený přímo pro mikrokontroléry Texas Instruments. Ten je vybaven vlastním compilerem a debuggerem a umožňuje využít všechny TI kity.

Další výhodou je volně dostupný balík zdrojových kódů a knihoven StellarisWare určený pro všechny Stellaris mikrokontroléry. Obsahuje knihovnu pro obsluhu veškerých vnitřních periferií, grafickou knihovnu pro grafické displeje, USB knihovnu, podporu Boot Loaderu a množství ukázkových kódů, díky kterým se programování stává jednodušším.

Rovněž datasheety obvodů jsou velmi podrobné a v případě nejasností je k dispozici oficiální TI internetové fórum pro Stellaris ARM Cortex-M mikrokontroléry [16], kde je možné problém prodiskutovat s odborníky.

4.2 Řídicí deska

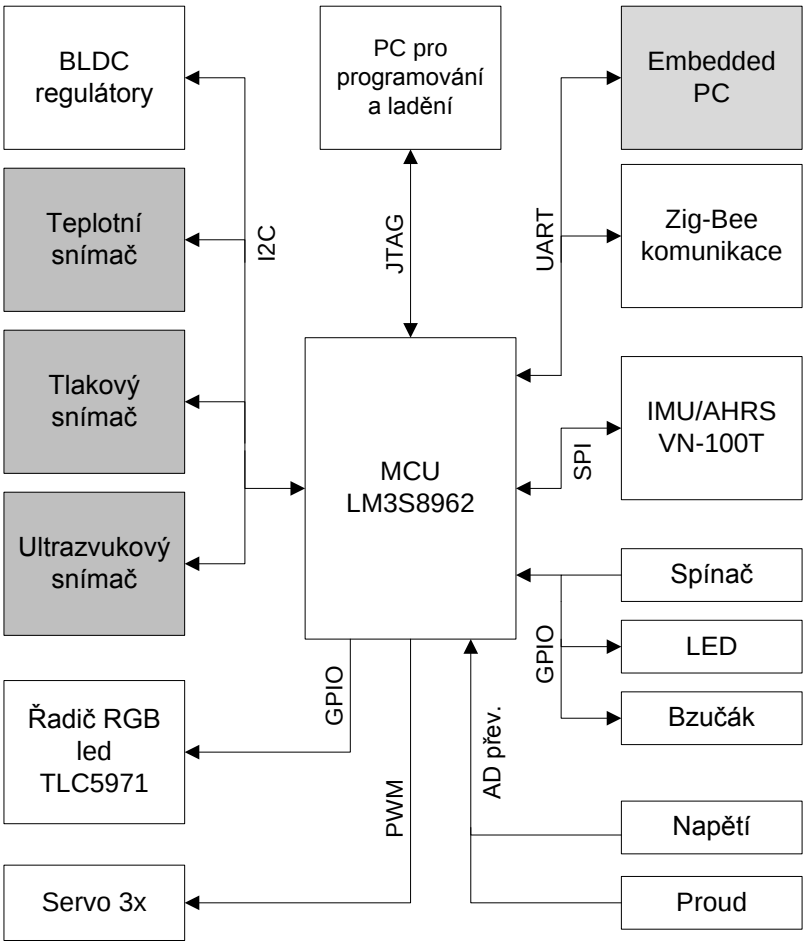
V první části vývoje byl systém testován na Evaluation Kitu EKS-LM3S8962, který je osazený výše popsaným mikrokontrolérem. Kit je vybaven některými základními periferiemi (tlačítka, LED diody, grafický displej, UART/USB převodníkem pro komunikaci s PC, komunikační konektory), které usnadňují ladění programu a zároveň slouží jako programátor obvodu s možností debuggeru.

Později, až byla k dispozici řídicí deska, vývoj a testování probíhalo na ni. Návrh zapojení a desky plošných spojů probíhal v těsné spolupráci s touto prací, podrobněji je však popsán v práci [6]. I přes to byla zachována zpětná kompatibilita s vývojovým kitem, který tak stále může sloužit k testování.

Schéma zapojení řídicí desky je v Příloze 2, na Obrázku 10 pak její blokové schéma z pohledu řídicího systému. Šedě označené bloky jsou předmětem vývoje práce [6] a budou v budoucnu zakomponovány do řídicího systému popisovaného v této práci.

K mikrokontroléru jsou pomocí různých rozhraní připojeny následující periferie: inerciální jednotka VN-100T pomocí SPI rozhraní, regulátory BLCD motorů, teplotní, tlakový a ultrazvukový snímač pomocí rozhraní I^2C, Zig-Bee modul pro komunikaci s pozemní stanicí a embedded PC pro zpracování dat z kamery jsou připojeny pomocí rozhraní UART. Dále je pomocí PWM signálů ovládána trojice polohových serv pro stabilizaci kamery. Pomocí AD převodníků jsou snímány analogové údaje o napětí baterie a odebíraném proudu. Řadič signalizačních RGB led diod komunikuje po vlastním sériovém protokolu, který je implementován na dvojici GPIO pinů. Na další tři GPIO piny je připojen spínač, signalizační led dioda a bzučák sloužící k ladění a ovládání. JTAG rozhraní slouží k programování flash paměti a ladění.

Následující podkapitoly se věnují snímači VN-100T, který přináší oproti řešení v první verzi robotu nové možnosti při zpracování dat z inerciálních snímačů. Dále pak RGB Led řadiči, který obstarává světelnou signalizaci robotu. Pro řízení BLDC motorů jsou použity stejné regulátory jako v první verzi [1], nebudou proto blíže popisovány. Dálkovému řízení a začlenění do nadřazeného systému je věnována kapitola Začlenění do systému Cassandra-WPF.



Obrázek 10: Blokové schéma řídicí desky

4.3 Snímač VectorNav VN-100T

Znalost orientace létajícího robotu je naprosto stěžejní, jelikož se jedná o vstupní údaje pro stabilizační regulátory. Orientace bývá nejčastěji popsána trojicí Eulerových úhlů vztažených k soustavě souřadnic, která je spojena se Zemí. V letecké technice jsou ustáleny výraz Yaw (rotace kolem svislé osy, neboli kurz), Pitch (sklon) a Roll (náklon). Tyto údaje je možné určit pomocí inerciálních snímačů, kterými jsou gyroskopy pro měření úhlové rychlosti, akcelerometry pro měření zrychlení a magnetometry pro měření magnetického pole. Který z těchto senzorů využít je otázkou konkrétní aplikace, ovšem nejlepších výsledků se dosáhne vhodnou kombinací všech tří. Nejedná se o jednoznačný proces, ale o poměrně komplikovaný, jelikož je závislý na mnoha okolních faktorech. Záleží například na přítomnosti rušivého magnetického pole, dynamického zrychlení (vibrací) a dynamice systému.

V první verzi robotu byla použita inerciální jednotka Analog Devices ADIS16405, jejíž výstupem jsou údaje o působících zrychleních, úhlových rychlostech a magnetickém poli. Údaje z gyroskopů a akcelerometrů byly vhodným způsobem

sloučeny tak, aby byly získány úhlové natočení pitch a roll. Algoritmus kompenzoval integrální chybu gyroskopů pomocí úhlových natočení vypočtených ze statického zrychlení. Podrobněji je popsán v práci [1]. Snímač VN-100T nabízí oproti ní již integrovanou datovou fúzi údajů ze všech inerciálních snímačů pomocí Kalmanova filtru.

Společnost VectorNav [17] je zaměřena na vývoj a výrobu přesných inerciálních jednotek a orientačních snímačů, nazývaných také AHRS (*Attitude Heading Reference System*). Samotné snímače fyzikálních veličin, které využívají, jsou na trhu volně dostupné. VectorNav se zabývá především pokročilým vyhodnocováním a zpracováním jejich signálů pro potřeby inerciální navigace. V Tabulce 2 jsou uvedeny základní parametry použitých snímačů.

Použitý snímač	Počet os	Dynam. rozsah	Rozlišení	Stabilita
Gyroskop InvenSense IDG-500	2	±500 °/s	< ±0,01 °/s	< ±0.06 @ 25°C
Gyroskop InvenSense ISZ-500	1	±500 °/s	< ±0,01 °/s	< ±0.06 @ 25°C
Akcelerometr ADXL325	3	±2 g ±6 g	X/Y < 0,4 mg Z < 1,0 mg	X/Y < 2 @ 25°C Z < 3 @ 25°C
Magnetometr Honeywell HMC6042	2	±6 Gauss	-	0.125 mGauss @ 25°C
Magnetometr Honeywell HMC1041Z	1	±6 Gauss	-	0.125 mGauss @ 25°C

Tabulka 2: Základní parametry snímačů jednotky VN-100T [18]

VN-100T v každém okamžiku vzorkování vypočítává aktuální orientaci senzoru v prostoru na základě údajů z každého inerciálního snímače. Výstupem tak můžou být přímo Eulerovy úhly natočení (Yaw, Pitch, Roll), Quaternion nebo rotační matice. Fůze dat ze snímačů probíhá v Kalmanově filtru, jehož parametry jsou uživatelsky konfigurovatelné, čímž lze senzor optimalizovat do konkrétní aplikace. Touto problematikou se zabývá následující kapitola. Udávané parametry údajů o orientaci jsou shrnuty v Tabulce 3.

	Rozsah	Přesnost	Rozlišení	Teplotní citlivost
Yaw	± 180°	< 2,0° @ 25°C	< 0,05°	< 0,005°
Pitch, Roll	± 180°	< 0,5° @ 25°C	< 0,05°	< 0,005°

Tabulka 3: Přesnost údajů o orientaci jednotky VN-100T [18]

Údaje o orientaci jsou dostupné s frekvencí maximálně 200 Hz, vnitřní vzorkovací perioda snímačů je ovšem 120 kHz z důvodu filtrace dat.

4.3.1 Komunikace

VN-100T komunikuje s okolím pomocí sériového SPI rozhraní při frekvenci hodinového signálu až 18 MHz, nebo pomocí asynchronního sériového rozhraní na rychlosti až 921600 baud/s. Použitá vývojová deska navíc umožňuje díky integrovanému převodníku komunikaci s PC pomocí USB. Navržený řídicí systém ovšem využívá rozhraní SPI, především díky vysoké přenosové rychlosti a snadné integraci do mikrokontroléru.

Jelikož je snímač VN-100T vždy v roli *slave*, je vybaven navíc pinem DR_INT, kterým signalizuje, že jsou k dispozici nová data. Zařízení typu *master*, v tomto případě LM3S8962, tak může reagovat formou externího přerušení a zaslat po SPI příslušný příkaz pro získání požadovaných dat. K dispozici je 32 registrů, které slouží jednak k získání informací o snímači, jeho konfiguraci a uchování naměřených dat.

Výrobce ke snímači poskytuje knihovny funkcí pro mikrokontroléry, psané v jazyce C. Díky nim lze jednoduše přistupovat ke všem registrům, konfigurovat je a vyčítat aktuální data. Navíc jsou v nich obsaženy funkce obsahující všechen potřebný matematický aparát pro použití snímače. Knihovna se skládá z přenositelných souborů nezávislých na použité architektuře a ze souborů `VN_user.c` a `VN_user.h`, které slouží k propojení s daným mikrokontrolérem. V nich je třeba definovat funkci pro ovládání SS (*Slave Select*), funkci pro odeslání a příjem jednoho bytu po SPI, a funkci realizující zpoždění 1 μs.

4.3.2 Uživatelská konfigurace

Aby dával senzor neustále pravdivý údaj o orientaci robotu v prostoru, je nutné brát zřetel na podmínky, kterým je senzor vystavován. Rušení, které na senzory působí, může být známé, jako například vibrace a elektromagnetické pole způsobené běžícími motory v robotu. V takovém případě lze průběžně měnit parametry Kalmanova filtru (dále KF), podle toho, kdy a jaké rušení nastává. V případě zmíněného motoru by se jednalo o snížení důvěry akcelerometrům a magnetometrům a naopak zvýšení důvěry gyroskopům. Pro takové situace je možné v KF nastavovat následujících 10 parametrů:

- Parametr 1: Gyro random walk variance
- Parametr 2-4: X-axis, Y-axis, Z-axis gyro noise variance
- Parametr 5-7: X-axis, Y-axis, Z-axis magnetometer noise variance
- Parametr 8-10: X-axis, Y-axis, Z-axis accelerometer noise variance

První parametr nastavuje vliv odchylky způsobené šumem signálu gyroskopu. Ostatními parametry je nastavována nejistota pro daný snímač a jeho osu. Parametr

může být vnímán také jako důvěryhodnost údajů z jednotlivých snímačů, přičemž čím nižší číslo je nastaveno, tím je danému snímači nastavena vyšší důvěryhodnost.

V některých případech ovšem nevíme kdy a jaké rušení nastane. Jedná se o měření vektoru zrychlení a magnetického pole. Pro účely získání orientace senzoru je potřeba měřit tyto vektory tak, jak se samy vyskytují, neboť mají konstantní velikost. Magnetometr měří v ideálním případě pouze magnetické pole Země, akcelerometr tíhové zrychlení. Tomu tak vždy není. Obzvláště v mobilní robotice bývá robot vystavován dynamickým zrychlením a magnetickým polím z různých zdrojů. Z toho důvodu je algoritmus KF doplněn o následující 4 aktivní parametry:

- Akt. parametr 1: Magnetic Disturbance Gain
- Akt. parametr 2: Acceleration Disturbance Gain
- Akt. parametr 3: Magnetic Disturbance Memory
- Akt. parametr 4: Acceleration Disturbance Memory

První dva parametry udávají zesílení odchylky od standardní velikosti vektoru magnetického pole a zrychlení. Parametr může nabývat hodnotu v intervalu 0 až 1, přičemž při hodnotě 0 je změna velikosti vektoru ignorována a může tak být určena chybná orientace. Druhé dva parametry určují velikost paměti pro tento jev, neboli dobu, po kterou nebude senzor věřit údajům po zjištění špatné velikosti vektoru (hodnota 0 udává 0s, hodnota 0,99 udává přibližně 1-2s).

U magnetometru je možnost ještě další konfigurace. Jelikož bývá senzor často uložen v blízkosti kovových nebo feromagnetických materiálů, motorů a baterií, bývá vystaven jejich parazitním magnetickým polím. Tyto materiály a objekty bývají pevně spojeny se senzorem, například na těle robotu a proto je jejich vliv po celou dobu provozu konstantní. Takové magnetické pole se dá eliminovat. VN-100T je vybaven registry pro hard/soft iron kompenzaci. Kompenzace je popsána rovnicí (1). Proměnné $\{MX, MY, MZ\}$ představují měřené magnetické pole, $\{X, Y, Z\}$ je magnetické pole po kompenzaci. Zbylých 12 parametrů je konfigurovatelných, ovšem jejich manuální zjištění je velmi komplikované. Automatický výpočet parametrů jde provést pomocí dodávaného softwaru Sensor Explorer.

$$\begin{Bmatrix} X \\ Y \\ Z \end{Bmatrix} = \begin{bmatrix} S11 & S12 & S13 \\ S21 & S22 & S23 \\ S31 & S32 & S33 \end{bmatrix} \cdot \begin{Bmatrix} MX - H1 \\ MY - H2 \\ MZ - H3 \end{Bmatrix} \tag{1}$$

Důležitou funkci plní příkaz *Tare*, který slouží k vytvoření souboru referenčních vektorů pro KF. Po dobu zhruba jedné sekundy jsou snímány údaje o magnetickém poli a zrychlení a dále je vypočtena průměrná hodnota, která je uložena jako referenční pro daný vektor. Také orientace je uvedena do původního stavu, což odpovídá nastavení nulových počátečních podmínek pro gyroskopy.

4.3.3 Testování

Způsob, jakým vyladit KF pro konkrétní aplikaci, popisuje dvojice aplikačních poznámek [19] a [20]. Jedná se o experimentální postup, který vychází z pochopení vlivů jednotlivých parametrů. Konkrétní konfigurace byly testovány přímo na těle robotu, aby tak mohly být eliminovány i parazitní vlivy. Především tak byl testován vliv vibrací způsobenými motory a vliv elektromagnetického pole.

V první řadě byly nastaveny referenční vektory gravitačního a magnetického pole. To bylo provedeno za situace, kdy snímač ležel ve vodorovné poloze. Příkazem *Tare* byly získány hodnoty uvedené v Tabulce 4. Pro porovnání jsou uvedeny také defaultní hodnoty. Především hodnoty magnetického pole se značně liší, jelikož jeho vektor je závislý na poloze na Zemi. Podobně je to s vektorem tíhového zrychlení, ten se již ovšem nemění tak významně. Z jeho hodnot je zároveň patrné, že snímač neležel v dokonale vodorovné poloze, jelikož se síla rozkládá také do osy X a Y.

Údaj	Naměřená hodnota	Defaultní hodnota
Reference mag. pole osa X	0,489150300	1,000000
Reference mag. pole osa Y	0,001712324	0,000000
Reference mag. pole osa Z	1,528319000	1,800000
Reference tíh. zrych. osa X	-0,07880652	0,000000
Reference tíh. zrych. osa Y	-0,03299210	0,000000
Reference tíh. zrych. osa Z	-9,94600900	-9,793746

Tabulka 4: Referenční vektory gravitačního a magnetického pole

Dále byla provedena hard/soft iron kalibrace magnetického snímače, aby bylo eliminováno rušivé magnetické pole těla robotu. To bylo provedeno programem Sensor Explorere v1.3.8.368. Tělem robotu s připevněným snímačem bylo otáčeno okolo všech os a postupně byly vypočítány kompenzační parametry. Výsledek kalibrace je obsažen v rovnici (2).

$$\begin{Bmatrix} X \\ Y \\ Z \end{Bmatrix} = \begin{bmatrix} +4{,}977842\text{E-}01 & -2{,}228533\text{E-}03 & -1{,}202732\text{E-}03 \\ -2{,}228533\text{E-}03 & +4{,}895870\text{E-}01 & +2{,}437800\text{E-}03 \\ -1{,}202732\text{E-}03 & +2{,}437800\text{E-}03 & +4{,}582553\text{E-}01 \end{bmatrix} \cdot \begin{Bmatrix} MX - (-6{,}647429\text{E-}01) \\ MY - (-6{,}620332\text{E-}02) \\ MZ - (-7{,}031716\text{E-}02) \end{Bmatrix} \tag{2}$$

Následně mohla být pomocí série testů zjištěna nejvhodnější konfigurace KF. Parametry byly nastavovány s ohledem na dynamiku robotu, aby hodnoty úhlových natočení co nejpřesněji korespondovaly s jeho pohyby. Praktické ověření bude

provedeno až na reálných testech hotového systému. Parametry KF jsou uvedeny v Tabulce 5.

Parametr	Hodnota
Angular Walk	1E-05
Úhlová rychlost osa X	1E-07
Úhlová rychlost osa Y	1E-07
Úhlová rychlost osa Z	1E-07
Magnetické pole osa X	0,001
Magnetické pole osa Y	0,001
Magnetické pole osa Z	1E-05
Zrychlení osa X	1E-05
Zrychlení osa Y	1E-05
Zrychlení osa Z	1E-05

Tabulka 5: Nastavené parametry Kalmanova filtru VN-100T

4.4 Komunikační modul

První verze robotu využívala k dálkovému řízení standardního modelářského vybavení pracující v pásmu 35 MHz, kde byly řídicí signály kódovány formou impulsů s proměnnou délkou. Tento způsob samozřejmě neumožňoval přenos jiných údajů. Proto byl nahrazen Zig-Bee moduly XBP24-DMSIT-250J od firmy Digi International, Inc. Jeden modul je připojen pomocí rozhraní UART k mikrokontroléru, druhý přes příslušný převodník k řídicímu PC. Tím je vytvořena fyzická vrstva komunikačního modelu. Moduly jsou konfigurovány na přenosovou rychlost 19200 baud/s, 8 datových bitů, 1 start bit, 1 stop bit, bez parity.

Samotné dálkové řízení probíhá pomocí systému Cassandra-WPF, s kterým robot obousměrně komunikuje. Tomu je věnována samostatná kapitola Začlenění do systému Cassandra-WPF.

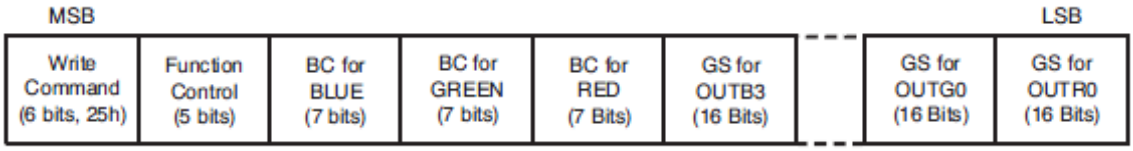
4.5 Řadič RGB led diod TLC5971

Na každém ze čtyř ramen robotu je umístěna RGB led dioda, sloužící k lepšímu rozpoznání robotu za letu. Při nočním letu je barevné značení přímo nezbytné. Volba barev je sice v tomto případě libovolná, ale je vhodné dodržovat standardní značení letadel. To vyžaduje zelené světlo na levém křídle, červené na pravém a bílé na ocasu. V případě quadrocopteru jsou za ocas považována dvě zadní ramena.

Ovládat trojici barev každé RGB led diody samostatným GPIO pinem mikrokontroléru by vyžadovalo celkem 12 pinů. Proto bylo zvoleno vhodnější řešení pomocí řadiče RGB led diod TLC5971 [21] od výrobce Texas Instruments. Jedná se o

12kanálový proudový zdroj, který dokáže měnit střídu výstupního PWM signálu v 65536 krocích (16 btiů) a poskytnout až 60 mA na kanál. S okolím obvod komunikuje pomocí dvouvodičového sériového rozhraní (DATA, CLK).

Jeden řídicí příkaz, zaslaný po sériovém rozhraní, se musí vždy skládat s posloupnosti 224 bitů. Posloupnost začíná 6ti bity s hodnotou 0x25, což je *Write Command*, následuje 5 konfiguračních bitů *Function Control*, dále 3x7bitů pro globální ovládání jasu každé barvy (*Brightness Control*), a na závěr 12x16bitů pro nastavení PWM střídy pro jednotlivé kanály (*Grayscale*). Posloupnost je znázorněna na Obrázku 11.



Obrázek 11: Složení řídicí posloupnosti TLC5971 [21]

Z pětice konfiguračních bitů je nejpodstatnější bit *BLANK*, který aktivuje proudové výstupy a bit *DSPRPT*, který zajistí *auto-refresh* výstupu. Díky němu stačí zaslat řídicí posloupnost pouze jedenkrát a hodnoty výstupů jsou ponechány nezměněny.

Pro snadné použití řadiče byla vytvořena jednoduchá knihovna funkcí v jazyce C pro použitý mikrokontrolér. Skládá se ze souborů `led_driv_TLC5971.c` a `led_driv_TLC5971.h`. Pro komunikaci s řadičem je využit pin PC6 pro DATA a pin PC7 pro CLK. První funkce nastavuje standardní barevné značení popsané výše:

```
void TLC5971SetStd(void)
```

Druhá funkce umožňuje nastavit barvy jednotlivých RGB led diod, přičemž hodnota parametru 0=červená, 1=zelená, 2=modrá, 3=bílá:

```
void TLC5971Set( long lLed1, long lLed2, long lLed3, long lLed4)
```

4.6 Stabilizátor kamery

Robot je vybaven kamerou určenou pro automatické přistávání, čímž se zabývá práce [6]. Aby byla detekce značky úspěšná, musí kamera směřovat stále kolmo k zemi, což lze zajistit pouze její aktivní stabilizací. Tato stabilizace musí zajistit, aby byla kamera natočena v ose X a Y vždy o stejný úhel, jako zaujímá tělo robotu v těchto osách k zemi, pouze s opačným znaménkem.

Aktivní prvky tvoří trojice modelářských serv, které jsou vhodně umístěny na jednoduché konzoli zavěšené pod tělem robotu, viz. Obrázek 12. Servo v ose Z není v případě stabilizace prozatím využito.

Obrázek 12: Aktivní stabilizátor kamery

Modelářská serva jsou vybavena zpětnovazební polohovou regulací, proto vstupní signál tvoří přímo požadované úhlové natočení osy motoru (standardně v rozmezí -90° až 90°). Vstupní informace je kódována pomocí střídy PWM signálu. Perioda signálu je běžně 20 ms, přičemž nulová výchylka osy odpovídá pulsu trvajícímu 1,5 ms. Pro generování těchto signálů jsou využity hardwarové PWM generátory, které má mikrokontrolér LM3S8962 implementovány. Jejich softwarová obsluha bude blíže popsána v kapitole Vytvoření FreeRTOS aplikace.

5 ZAČLENĚNÍ DO SYSTÉMU CASSANDRA-WPF

Změnou oproti první verzi robotu prošel také způsob dálkového řízení. Předcházející kapitola mimo jiné popisuje hardware použitý ke komunikaci, který využívá Zig-Bee modulů. Datová komunikace tak nově umožňuje kromě řídicích pokynů přenášet libovolná data i v opačném směru, jimiž mohou být letové údaje, stav hardwaru, či data ze snímačů. Pozemní stanice, tvořena PC, je vybavena řídicím systémem Cassandra-WPF, který je vyvíjen ústavu Automatizace a měřící techniky VUT v Brně. Kapitola se blíže věnuje tomuto systému a jeho možnostem a dále způsobu komunikace s ním.

5.1 Cassandra-WPF

Cassandra-WPF je real-time řídicí systém pro průzkum neznámého prostředí pomocí skupiny různorodých robotů. Představuje univerzální uživatelské rozhraní, které umožňuje ovládat různé roboty velmi podobným způsobem. Cílem je mít skupinu průzkumných robotů různých velikostí, konstrukcí, konfigurací a vybaveností snímačů schopnou plnit různé průzkumné mise. Běžná konfigurace se skládá ze skupiny pozemních mobilních robotů, která by v případě quadrocopteru byla doplněna o podporu létajícího robotu. Na Obrázku 13 je ukázka uživatelského rozhraní.

Obrázek 13: Uživatelské rozhraní Cassandra-WPF [24].

Systém plně podporuje tzv. vizuální teleprezenci. To umožňuje ovládat jednu z kamer robotu pomocí pohybů hlavy operátora. Podporuje také různé ovládací zařízení HID, jako jsou gamepady a joysticky, usnadňující dálkové ovládání robotu. Systém

Cassandra-WPF je naprogramován v Microsoft Visual Studiu 2010 s využitím programovacího jazyka C# a .NET technologie.

Cílem této práce je vytvoření řídicího systému pro létající robot Uranus, který bude do systému Cassandra-WPF v budoucnu začleněn. Řídicí systém musí být schopen s ním komunikovat, a to buď pomocí standardního sériového rozhraní, nebo pomocí internetového UDP protokolu. Komunikaci jsou věnovány následující kapitoly.

5.2 Model komunikace

Tato podkapitola popisuje komunikaci robota s řídicím systémem na referenčním ISO-OSI modelu (Tabulka 6) [23]. Z hlediska přenosu dat jsou zde nejdůležitější spodní dvě vrstvy, patřící mezi tzv. *Media layers*. Horní 4 vrstvy, *Host layers*, zde většinou nejsou implementovány.

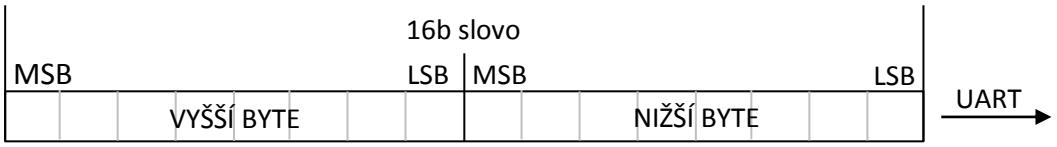
Vrstva	Název
7	Aplikační
6	Prezentační
5	Relační
4	Transportní
3	Síťová
2	Spojová
1	Fyzická

Tabulka 6: Vrstvy komunikačního ISO-OSI modelu

Fyzická vrstva – tato vrstva popisuje fyzický přenos dat. Již bylo zmíněno, že ke komunikaci slouží bezdrátové moduly Zig-Bee pracující v pásmu 2,4 GHz. Na straně robotu je modul připojen dvojvodičově asynchronním sériovým rozhraním UART k mikrokontroléru LM3S8962, pracující na 3,3 V logice. Druhý modul je připojen k PC pomocí převodníku RS232RL. Dále tato vrstva specifikuje složení přenášeného bytu a přenosovou rychlost. Byte se skládá z jednoho start bitu, 8 datových bitů a jednoho stop bitu, parita využita není. Přenosová rychlost je nastavena na 19200 bps.

Spojová vrstva – v této vrstvě je definováno pořadí bitů v bytu, pořadí bytů v případě více bytového slova a samotný formát datového rámce (paketu).

Při odesílání bytu na sériové rozhraní je nejdříve odeslán nejméně významný bit LSB a MSB jako poslední. Stejná logika je dodržena při odesílání slova, nejdříve nejméně významný byte a na závěr nejvíce významný byte. Na Obrázku 14 je příklad odeslání 16b slova.



Obrázek 14: Pořadí bitů a bytů při odesílání dat

Struktura datového rámce vychází ze struktury rámců určených pro komunikaci s digitálními servy Dynamixel od firmy Robotis [22]. To je z důvodu již částečné implementace protokolu do systému Cassandra-WPF. Struktura jednoho rámce je znázorněna v Tabulce 7.

Pořadí	Popis	Velikost v bytech
1	Start byte 1 (0xFF)	1
2	Start byte 2 (0xFF)	1
3	ID zařízení (zprávy)	1
4	Délka rámce	1
5	Instrukce	1
6 až 6+(N-1)	Parametry	1 až N
6+N	Kontrolní součet	1

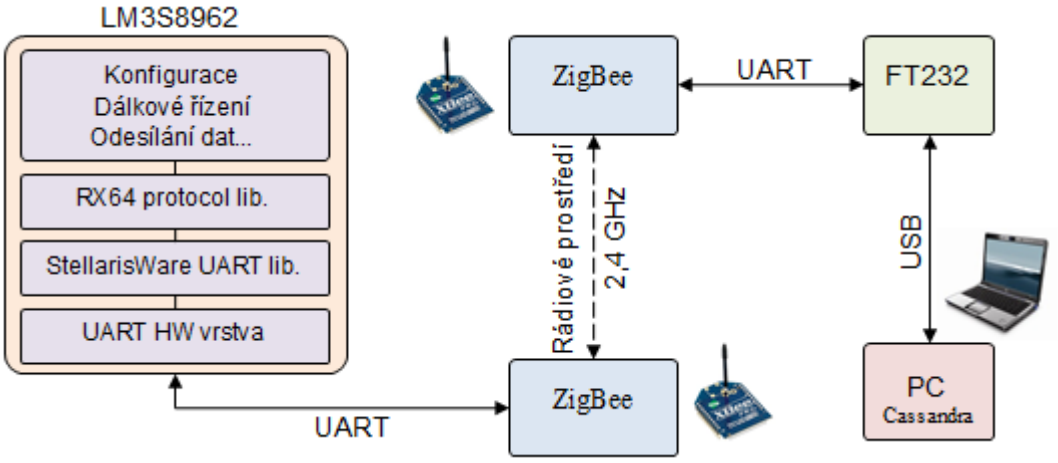
Tabulka 7: Struktura datového rámce Dynamixel [22]

Rámec začíná dvojicí bytů s konstantní hodnotou 0xFF. ID zařízení v této práci není potřeba, jelikož komunikace probíhá vždy s jedním robotem. Tento byte je využit pro rozlišení typu zprávy, proto bude dále označován jako ID zprávy. Délka datového rámce je určena počtem parametrů + 2. Byte instrukce prozatím nebyl využit. Následuje libovolný počet parametrů, které představují přenášená data. Všechny přenášené zprávy v této práci (Tabulka 8) mají konstantní počet parametrů dva. Posledním bytem je kontrolní součet (*checksum*), který slouží k ověření správnosti přijatého rámce. Je vypočítán dle vzorce (3).

$$checksum = \neg(ID + délka + instrukce + param.1 + ... + param.N) \tag{3}$$

Pokud vypočtená hodnota přesáhne 255, je použit nižší byte. Znak $\neg$ představuje negaci.

Datový rámec je odesílán vzestupně podle pořadí bytů, tj. jako první je odeslán Start byte 1.



Obrázek 15: Ilustrace komunikačního modelu

Síťová vrstva v této práci nemá opodstatnění, jelikož se jedná o přímou komunikaci dvou zařízení. ***Transportní vrstva***, která běžně zajišťuje přenos dat mezi koncovými uzly, lze přirovnat k UDP protokolu. Pokud některé datové rámce nejsou doručeny, nebo jsou chybné, již s nimi není počítáno a není požadováno opětovné zaslání. To by způsobovalo nechtěné prodlevy při řízení. Zbylé vrstvy modelu nejsou implementovány a jejich funkce jsou případně řešeny individuálně dle potřeby aplikace. Komunikační model ilustruje Obrázek 15.

5.3 Implementace do systému

Pro implementaci výše popsaného komunikačního protokolu do řídicího systému bylo třeba vytvořit soubor funkcí pro mikrokontrolér LM3S8962. Ve zdrojových souborech `RX64_protocol.c` a `RX64_protocol.h` jsou implementovány funkce pro příjem, vytvoření a odeslání paketu.

Funkce pro příjem přebírá parametr o cílovém paketu typu `RX64PACKET` a číslu použitého UART rozhraní. Její hlavička je následovná:

```
char RX64ReceivePacket(RX64PACKET* p_rx64NewRecPacket, unsigned long ulPortNum)
```

Funkce pro vytvoření paketu má za parametry data paketu (ID, instrukce, pole parametrů, počet parametrů) a ukazatel na strukturu `RX64PACKET`, kam bude uložen. Její hlavička je následovná:

```
void RX64CreatePacket(unsigned char id, unsigned char instr, unsigned char* param, unsigned char numOfParam, RX64PACKET* p_rx64NewPacket)
```

Další funkce slouží pro odeslání paketu. Jejími parametry je ukazatel na vytvořený paket a číslo UART rozhraní, po kterém bude odeslán. Její hlavička je následovná:

```
void RX64TransmitPacket(RX64PACKET* p_rx64TransPacket, unsigned long ulPortNum)
```

Poslední funkce slouží k dekódování paketu dle ID zprávy. Výsledky dekódování jsou uloženy do proměnných typu `p_myJoystick` a `p_myConfig`.

```
void RX64DecodePacket(RX64PACKET* p_rx64NewRecPacket, JOYSTICK* p_myJoystick, CONFIG* p_myConfig)
```

5.4 Přenášená data

V současné době je komunikace využívána především k potřebám dálkového řízení robotu. Je-li aktivní *run mode*, robotu je cyklicky odesílána pětice paketů (datových rámců). První paket přenáší zprávu *engines enable* s hodnotou 1, další čtyři pakety obsahují údaje o výchylkách ovládacích pák gamepadu. Při aktivaci *stop mode* je jednorázově odeslána stejná pětice paketů, ovšem s hodnotou *engines enable* 0 a defaultními hodnotami výchylek pák.

Robot během svého provozu odesílá také některé data do PC. Kompletní přehled implementovaných zpráv je v Tabulce 8.

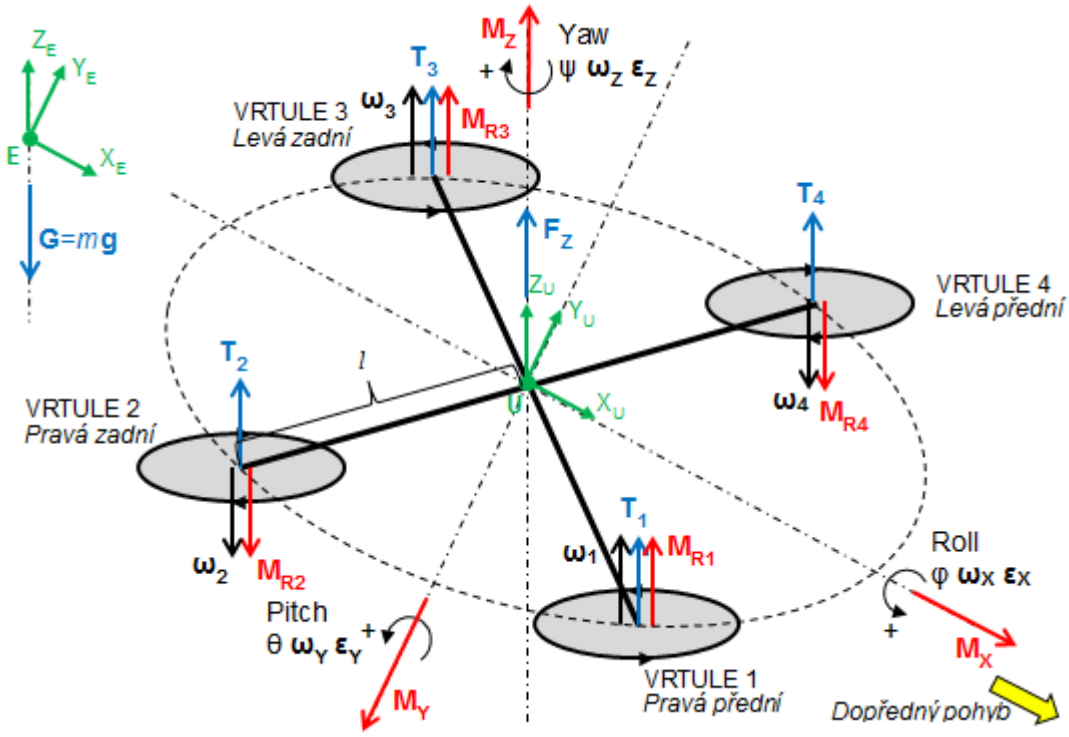
ID zprávy (hexa)	Popis	Rozsah hodnot	Defaultní (stop mode) hodnota
Cassandra → Robot			
0x10	Výchylka pitch ovl.	0x0000 – 0x3FFF	0x0200
0x11	Výchylka roll ovl.	0x0000 – 0x3FFF	0x0200
0x12	Výchylka thrust ovl.	0x0000 – 0x3FFF	0x0200
0x13	Výchylka yaw ovl.	0x0000 – 0x3FFF	0x0200
0x14	Engines enable	0x0000, 0x0001	0x0000
Robot → Cassandra			
0x20	Napětí baterie	0x0000 – 0xFFFF	-
0x21	Náklon *roll*	0x0000 – 0xFFFF	-
0x22	Sklon *pitch*	0x0000 – 0xFFFF	-
0x23	Kurz *yaw*	0x0000 – 0xFFFF	-

Tabulka 8: Implementované zprávy pro komunikaci robotu s Cassandrou

6 IDENTIFIKACE SYSTÉMU A NÁVRH STABILIZACE

Létající roboti jsou obecně z hlediska řízení mnohem komplikovanější, než roboti určení pro pohyb po zemi, což dáno naprostou volností pohybu v prostoru. Především pak konstrukce vrtulníkového typu, ať už jedno, či více rotorové, se neobejdou bez jisté úrovně automatické stabilizace. Ta má za úkol zajistit, aby byl stroj pro člověka ovladatelný, nebo dokonce plně autonomní. Problematika vychází z pochopení chování robotu a sestavení jeho matematického modelu. Vytvořený model je vždy jen určitou aproximací chování robotu v reálném světě a jeho přesnost je volena dle konkrétních požadavků aplikace.

Tato kapitola se zabývá identifikací základních parametrů nové konstrukce robotu, s jejichž pomocí je sestaven matematický model. Ten je dále využit pro návrh regulátorů stabilizujících rotační pohyby robotu a následně je provedena jejich implementace a otestována funkčnost. Celá problematika je zde zjednodušena a popsána v jedné kapitole, jelikož hlavním cílem je vytvoření a otestování řídicího systému. Podrobněji se tvorbou modelu quadrocopteru zabývá například [25], nebo diplomové práce [4] a [5], které dále popisují implementaci stavového regulátoru.



Obrázek 16: Kinematické schéma robotu

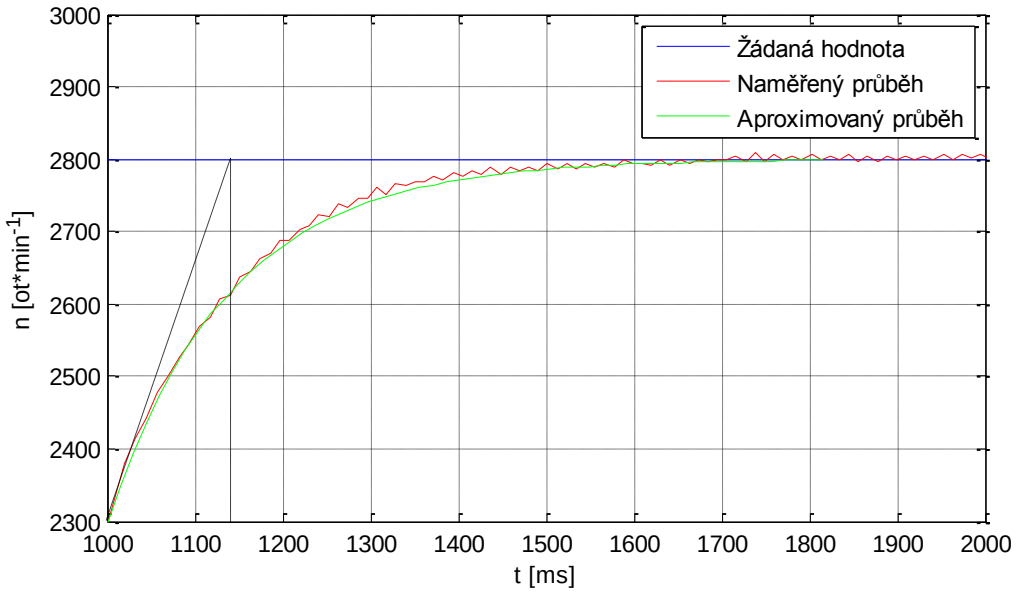
6.1 Identifikace a vytvoření modelu

Před samotným matematickým popisem musí být definovány veškeré potřebné konstanty robotu (rozměry, hmotnost, momenty setrvačnosti atd.) a vektory (síly, momenty síly, otáčky atd.). To vše ilustruje Obrázek 16 s ohledem na aktuální konstrukci robotu.

Na obrázku je znázorněna dvojice souřadnicových systémů, systém $\mathbf{E\{X_E,Y_E,Z_E\}}$ je pevně spojen se zemí, systém $\mathbf{U\{X_U,Y_U,Z_U\}}$ je spojen s konstrukcí robotu. Konstrukce se skládá ze čtyř motorů s vrtulemi umístěnými na konci ramen délky *l*. Každá vrtule, otáčející se rychlostí $\boldsymbol{\omega_n}$, způsobuje sílu $\mathbf{T_n}$ v kladném směru osy $\mathbf{Z_U}$ a moment síly $\mathbf{M_{Rn}}$, jehož směr závisí na směru otáčení vrtulí. Vrtule 1 a 3 mají rychlost kladnou, vrtule 2 a 4 zápornou. To je z důvodu vyrovnání jejich reakčních momentů. Osa $\mathbf{X_U}$ robotu má kladnou orientaci mezi dvojicí předních vrtulí. Úhlové natočení kolem této osy je způsobeno momentem $\mathbf{M_X}$ a je označováno jako *Roll*, nebo φ. Dále je definována úhlová rychlost $\boldsymbol{\omega_X}$. Obdobné platí pro osy robotu $\mathbf{Y_U}$ a $\mathbf{Z_U}$.

6.1.1 Identifikace akčního členu

V první řadě byla pozornost věnována systému akčního členu. Ten je z hlediska řízení tvořen regulátorem, BLDC motorem a vrtulí. Žádaná vstupní hodnota otáček se po chvíli objeví i na výstupu, dá se proto předpokládat, že se jedná o setrvačný článek. Pomocí optické závory bylo proměřeno několik přechodových charakteristik pro kladnou a zápornou změnu žádaných otáček. Jedna z nich je na Obrázku 17. Grafickou metodou byla odečtena časová konstanta a průběh byl aproximován setrvačným článkem prvního řádu (4).



Obrázek 17: Přechodová charakteristika akčního členu

$$F_{MOT}(p) = \frac{1}{T_{MOT}\,p + 1} = \frac{1}{0.14\,p + 1} \tag{4}$$

Všechny naměřené průběhy jsou v Příloze 3. Již od pohledu je zřejmé, že jejich časové konstanty se liší. V Tabulce 9 je shrnutí získaných časových konstant.

Jednotkový skok	**Rozmezí otáček [ot./min.]**	**Časová konstanta T_{MOT} [s]**
Kladná změna	1223 - 1532	0,480
	1223 - 2293	0,160
	1223 - 3463	0,145
	2294 - 2555	0,145
	2294 - 2800	0,140
Záporná změna	1529 – 1219	0,670
	2290 – 1225	0,600
	3481 - 1219	0,520
	2549 – 2289	0,210
	2770 - 2277	0,250

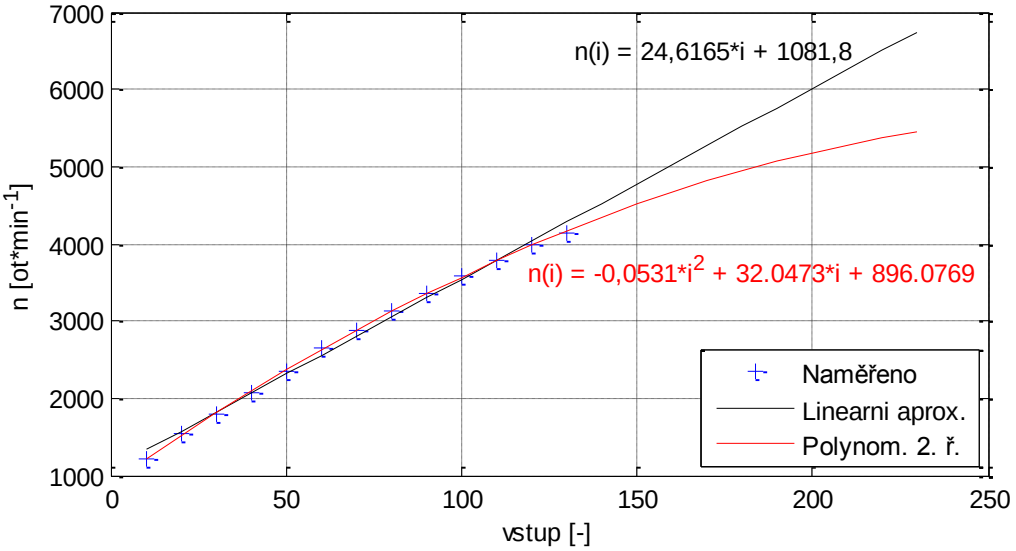
Tabulka 9: Naměřené hodnoty časových konstant akčního členu

Velikost časové konstanty závisí jednak na otáčkách motoru a také na polaritě skokové změny. První nelinearita se dá opomenout díky tomu, že otáčky vrtulí se budou vždy pohybovat ve stejné pracovní oblasti, která vychází z hmotnosti robotu. Tato pracovní oblast je v okolí 2200 - 2600 ot./min. Závislost časové konstanty na polaritě skokové změny vychází z faktu, že kladná změna otáček probíhá aktivně (energie je do motoru dodávána) a záporná změna pasivně (motor brzdí vlivem odporu vzduchu vrtule). Díky této nelinearitě by se model stal mnohem složitější, proto je opomenuta a časová konstanta je zvolena jako kompromis. Výsledný přenos akčního členu odpovídá (5). K podobným poznatkům došla také práce [26].

$$F_{MOT}(p) = \frac{1}{0.2\,p + 1} \tag{5}$$

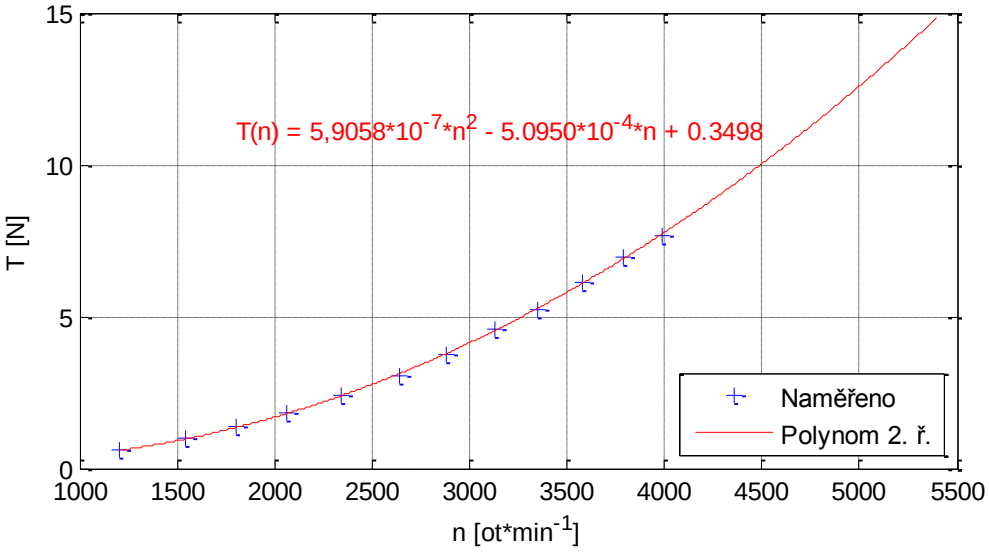
Jako řídicí hodnota pro nastavení žádaných otáček slouží 8bitové číslo s rozmezím 0-230. Závislost otáček se ovšem liší dle typu použitého motoru a vrtule, proto byla tato převodní charakteristika proměřena. Jelikož mají použité regulátory otáček proudové omezení na 10 A, nemohla být charakteristika proměřena celá, ale pouze po vstupní hodnotu 130. Maximální hodnota otáček je 4100 ot/min, minimální 1200 ot/min.

Naměřené data byla pro budoucí použití aproximována lineárně a polynomem druhého řádu. Průběhy jsou na Obrázku 18.

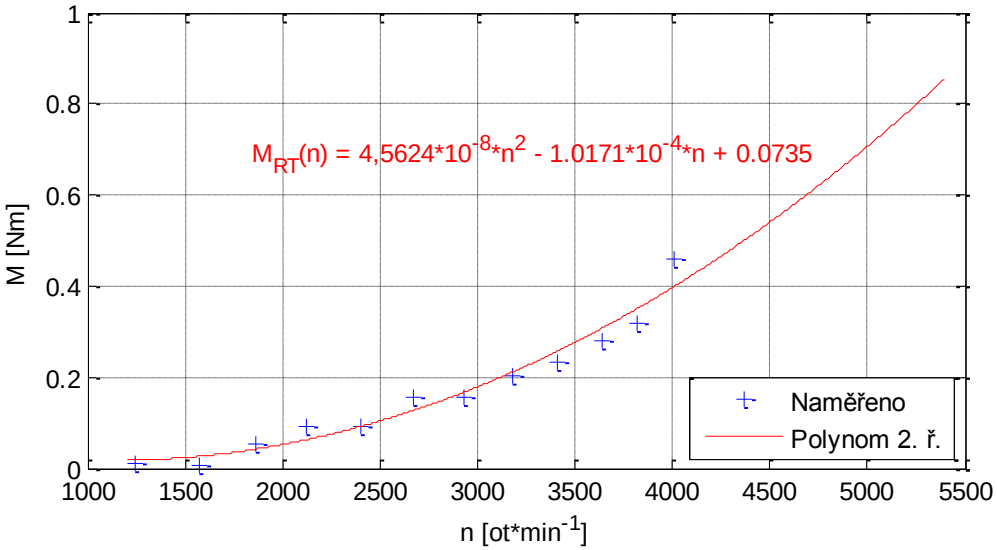


Obrázek 18: Převodní charakteristika vstupního údaje regulátoru na otáčky vrtule

Dále byly proměřeny závislosti tahu **T** akčního členu a reakčního momentu $\mathbf{M_R}$ na otáčkách vrtule. Tyto převodní charakteristiky jsou potřebné pro výpočet odpovídajícího akčního zásahu, obě byly proto aproximovány polynomem druhého řádu. Z důvodu výše zmíněného omezení otáček je možné měnit tah v rozmezí 0,6 až 8,0 N a reakční moment v rozmezí 0,01 až 0,41 Nm. Průběhy jsou na Obrázku 19 a Obrázku 20.



Obrázek 19: Převodní charakteristika otáček vrtule na tah



Obrázek 20: Převodní charakteristika otáček vrtule na reakční moment

6.1.2 Identifikace konstant

Dále je pro potřeby matematického modelu třeba určit několik základních konstant konstrukce robotu. Patří mezi ně hmotnost m, délka ramena l a momenty setrvačnosti J_X, J_Y, J_Z vztažených k osám robotu.

Hmotnost aktuální verze robotu činí 1,5 kg, ovšem hodnota se v závislosti na konfiguraci mění. Délka ramena je konstantní 0,365 m.

Momenty setrvačnosti se oproti předchozím údajům měřením určují velmi obtížně. Mnohem jednodušší je určitě přibližné hodnoty výpočtem. K tomu účelu lze využít odvozené vztahy pro výpočet momentu setrvačnosti homogenních těles [27]. Například obruč otáčející se okolo geometrické osy (6), tenká tyč otáčející se kolem osy vedené jejím středem kolmo k její délce (7) a vedené jedním z jejích konců kolmo k její délce (8) atd.

$$J = m \cdot l^2 \tag{6}$$

$$J = \frac{1}{12} m \cdot l^2 \tag{7}$$

$$J = \frac{1}{3} m \cdot l^2 \tag{8}$$

Ze všech uvedených vztahů vyplývá, že největší vliv na moment setrvačnosti mají prvky konstrukce nejvíce vzdálené od středu souřadnicového systému robotu. Zároveň tak prvky nejtěžší. Proto byly určeny momenty setrvačností způsobené motory s vrtulemi, rameny robotu a podvozku robotu. Výsledné hodnoty jsou v Tabulce 10.

Parametr	Hodnota [kg·m^2]
J_X	0,03898
J_Y	0,03775
J_Z	0,07229

Tabulka 10: Momenty setrvačnosti konstrukce robotu

6.1.3 Sestavení modelu

Jelikož je cílem vytvoření stabilizace rotačních pohybů robotu, může se na něj pohlížet jako na vícerozměrný systém, jehož vstupy tvoří čtveřice sil $\mathbf{T_n}$ (tahy motorů) a výstupy trojice momentů sil $\mathbf{M_X}$, $\mathbf{M_Y}$ a $\mathbf{M_Z}$. Snadno si lze také představit, že každý výstup je závislý na každém vstupu. Tyto závislosti popisuje rovnice (9). Záměrně byl přidán ještě čtvrtý výstup **T**, který představuje celkový tah motorů.

$$\begin{pmatrix} \mathbf{T} \\ \mathbf{M_X} \\ \mathbf{M_Y} \\ \mathbf{M_Z} \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ -0,707 \cdot l & -0,707 \cdot l & 0,707 \cdot l & 0,707 \cdot l \\ 0,707 \cdot l & -0,707 \cdot l & -0,707 \cdot l & 0,707 \cdot l \\ f_{TM} & -f_{TM} & f_{TM} & -f_{TM} \end{pmatrix} \cdot \begin{pmatrix} \mathbf{T_1} \\ \mathbf{T_2} \\ \mathbf{T_3} \\ \mathbf{T_4} \end{pmatrix} = \mathbf{R_{TM}} \cdot \begin{pmatrix} \mathbf{T_1} \\ \mathbf{T_2} \\ \mathbf{T_3} \\ \mathbf{T_4} \end{pmatrix} \tag{9}$$

Momenty $\mathbf{M_X}$ a $\mathbf{M_Y}$ vychází z velikosti síly $\mathbf{T_n}$ daného motoru a vzdálenosti od osy otáčení. Tato vzdálenost je dána 0,707 násobkem délky ramena, jelikož jsou osy robotu od os ramen pootočeny o 45° okolo osy $\mathbf{Z_U}$. Moment $\mathbf{M_Z}$ je dán součtem reakčních momentů vrtulí, funkce f_{TM} proto představuje převodní charakteristiku tahu vrtule na reakční moment.

Známe-li již moment síly působící v dané ose, je možné vypočíst úhlové zrychlení. To je dáno vztahem (10).

$$\varepsilon = \frac{\mathbf{M}}{J} \tag{10}$$

Dále je známo, že úhlové zrychlení je derivací úhlové rychlosti podle času, a stejně tak úhlová rychlost je derivací úhlového natočení. To popisují stavové rovnice (11) a (12).

$$\begin{aligned} \dot{\varphi} &= \omega_X \\ \dot{\theta} &= \omega_Y \\ \dot{\psi} &= \omega_Z \end{aligned} \tag{11}$$

$$\dot{\omega}_X = \varepsilon_X = \frac{\mathbf{M_X}}{J_X}$$
$$\dot{\omega}_Y = \varepsilon_Y = \frac{\mathbf{M_Y}}{J_Y} \tag{12}$$
$$\dot{\omega}_Z = \varepsilon_Z = \frac{\mathbf{M_Z}}{J_Z}$$

Nyní lze spojit popsané chování robotu a akčního členu do jednoho přenosu. Získá se tím trojice nezávislých přenosů (13), pro každou osu robota jeden. Vstup je z fyzikálního hlediska tvořen momentem síly a výstupem je úhlové natočení.

$$F_X(p) = \frac{\frac{1}{J_X}}{p^2(T_{MOT}p+1)} = \frac{25,65418}{p^2(0,2p+1)}$$
$$F_Y(p) = \frac{\frac{1}{J_Y}}{p^2(T_{MOT}p+1)} = \frac{26,49007}{p^2(0,2p+1)} \tag{13}$$
$$F_Z(p) = \frac{\frac{1}{J_Z}}{p^2(T_{MOT}p+1)} = \frac{13,83317}{p^2(0,2p+1)}$$

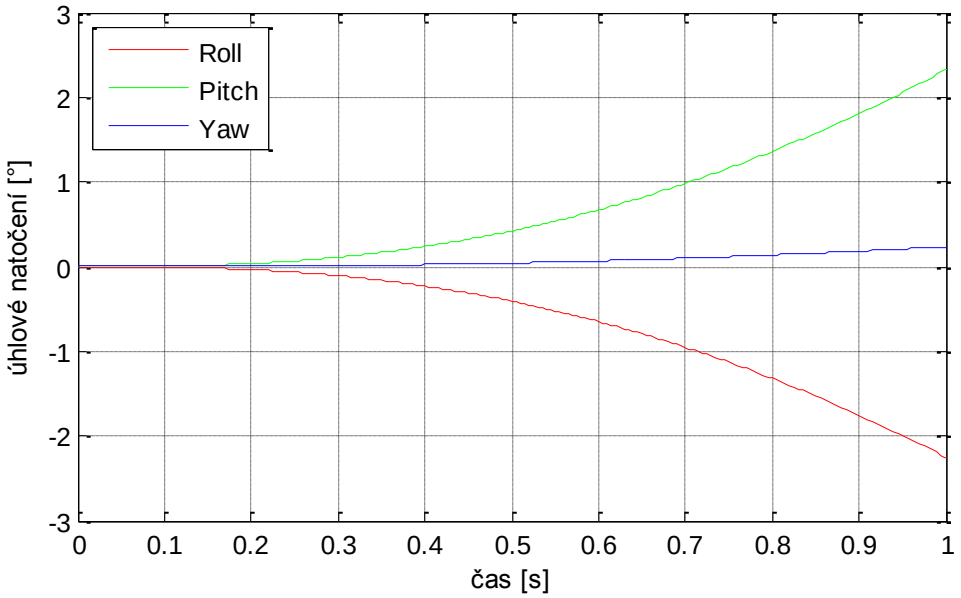
Realizovaný model představuje nejjednodušší aproximaci reálného systému robotu. Některé práce, například [25], tento model doplňují o gyroskopický moment těla robotu a vrtulí. Gyroskopický moment způsobený rotací hmotné vrtule jistě není zanedbatelný, ale jelikož by do stavových rovnic zanesl křížové vazby, které by dále zkomplikovaly návrh regulátoru, je z modelu vypuštěn. To lze obhájit také faktem, že se nepočítá s velkou dynamikou pohybů robotu, ale především s pracovní oblastí blízkou visu. Díky nízkým hodnotám úhlových rychlostí se gyroskopické momenty projeví pouze minimálně.

Přenosy (13) představují trojici lineárních systémů. V reálném světě se ovšem vždy setkáváme s jistými omezeními, především omezením akční veličiny. Tak je tomu i v našem případě, kdy akční člen je tvořen systémem regulátoru otáček, motorem a vrtulí. Díky proudovému omezení regulátoru jsou omezeny také otáčky motoru a vrtule a tím je omezen i tah. Jedná se o nelinearitu typu nasycení, která má v našem případě spodní mez 0,6 N a horní 8 N.

V prostředí Matlab-Simulink byl vytvořen model realizující výše popsané chování robotu. Jeho zapojení je v Příloze 4. Nutno podotknout, že z důvodu budoucí realizace, kde se počítá se vzorkovací frekvencí 200 Hz, byl model doplněn o tvarovače nultého řádu s periodou 5 ms.

6.2 Návrh regulátorů

Pomocí sestaveného modelu je možné ověřit chování systému. Na Obrázku 21 je odezva systému na skok řídicí veličiny T_1. Z průběhů jde vidět, že změna tahu vrtule 1 se projeví na úhlová natočení kolem všech tří os robotu. Následný návrh regulátorů bude ovšem proveden pro každou z os odděleně, jak popisují rovnice (13). Akční zásahy, které jsou tvořeny momenty síly **M**, budou následně přepočteny na akční zásahy v podobě tahů vrtulí dle rovnice (9).



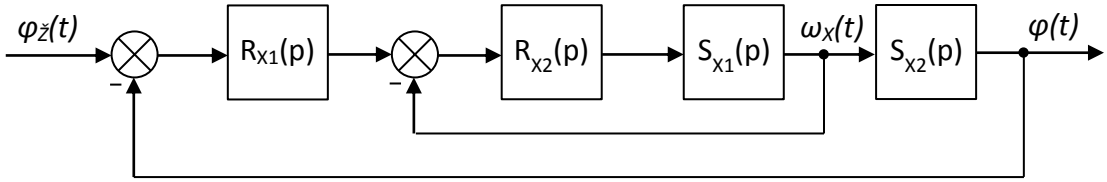
Obrázek 21: Odezva systému na jednotkový skok veličiny T_1

Všechny uvedené soustavy jsou nestabilní, jelikož kromě setrvačného článku obsahují dvojici astatismů. Se soustavami tohoto typu se v praxi běžně setkáváme ve formě polohových servomechanismů, kde akčním členem je servomotor a žádanou hodnotu tvoří poloha či úhlové natočení. Akční člen přitom odpovídá integračnímu článku se setrvačností. Soustava proto bývá rozdělena na dvě části a regulována je zvlášť úhlová rychlost servomotoru ve vnitřní smyčce a regulace na polohu ve vnější smyčce. Zavedena může být ještě další vnitřní regulační smyčka s proudovou zpětnou vazbou. Obecně se tato struktura nazývá jako regulační obvody s pomocnou regulovanou veličinou [28].

Soustava $F_X(p)$ může být podobným způsobem rozdělena na soustavy $S_{X1}(p)$ a $S_{X2}(p)$ (14).

$$F_X(p) = S_{X1}(p) \cdot S_{X2}(p) = \frac{\frac{1}{J_X}}{p(T_{MOT}p+1)} \cdot \frac{1}{p} = \frac{25,65418}{p(0,2p+1)} \cdot \frac{1}{p} \qquad (14)$$

Regulační obvod bude obsahovat dvojici regulátorů, $R_{X1}(p)$ pro vnější polohovou smyčku a $R_{X2}(p)$ pro vnitřní smyčku úhlové rychlosti. Struktura celého regulačního obvodu je na Obrázku 22.



Obrázek 22: Struktura regulačního obvodu s pomocnou regulovanou veličinou

Při návrhu regulátorů takového typu rozvětveného obvodu se postupuje od vnitřní smyčky po vnější. Vnitřní smyčky se obvykle volí tak, aby odpovídaly statickým soustavám na mezi aperiodicity. Pomalý přetlumený přechodový děj by se projevil jako velká časová konstanta v celkovém přenosu a zpomaloval by regulaci hlavní regulované veličiny.

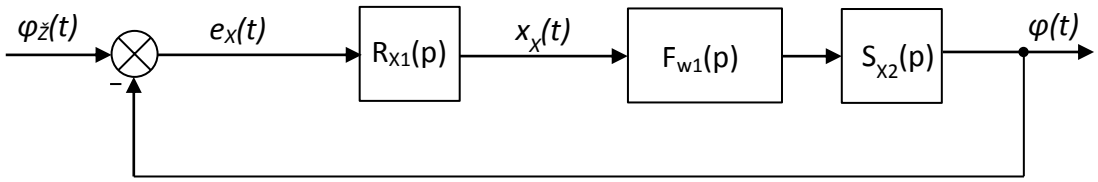
Aby se vnitřní uzavřená smyčka chovala jako statický systém, stačí, aby byl regulátor $R_{X2}(p)$ typu P (15). Přenos vnitřní uzavřené smyčky pak bude odpovídat setrvačnému článku druhého řádu (16).

$$R_{x2}(p) = k_2 \tag{15}$$

$$F_{w1}(p) = \frac{S_{X1}(p) \cdot R_{X2}(p)}{1 + S_{X1}(p) \cdot R_{X2}(p)} = \frac{k_2}{J_x T_{MOT} p^2 + J_x p + k_1} = \frac{k_2}{0,00796 p^2 + 0,03898 p + k_2} \tag{16}$$

Akční zásah regulátoru vnější smyčky, $R_{X1}(p)$, tvoří žádanou hodnotu pro systém vnitřní smyčky $F_{w1}(p)$ doplněný o přenos soustavy $S_{X2}(p)$, která je tvořena integrátorem (Obrázek 23). Díky integrátoru v soustavě je splněna podmínka pro jednotkový přenos řízení a regulátor typu P by tak umožňoval dosáhnout nulové ustálené odchylky. V reálném světě ovšem zcela jistě bude na vstup soustavy vstupovat chyba, která může být způsobena nevyvážeností konstrukce, poryvy větru apod. V této situace musí integrační složku obsahovat regulátor, musí být proto typu I, PI, nebo PID. Pro další návrh bude počítáno s regulátorem PID, který nabízí nejvíce možností nastavení. Jeho přenos je dán (17).

$$R_{x1}(p) = r_0 + \frac{r_i}{p} + r_d p = K_R (1 + \frac{1}{T_I p} + T_D p) \tag{17}$$



Obrázek 23: Upravená struktura regulačního obvodu

Výše uvedené platí pouze teoreticky, již v předchozí kapitole byla řeč o jistých rozdílech chování systému v reálném světě. Řeč je zejména o nelinearitě typu nasycení, která omezuje velikost akčního zásahu. Další rozdíl vychází z diskrétní realizace regulátorů, ale i snímačů. Snímače, tvořící zpětné vazby od výstupu soustavy, poskytují hodnotu v diskrétních intervalech 5 ms. Proto byly do zpětných vazeb vloženy přidržovače nultého řádu s touto periodou. Jejich použití má za následek vznik dopravního zpoždění o velikosti poloviny vzorkovací periody, tj. 2,5 ms. Regulátory pracují se stejnou vzorkovací periodou 5 ms.

Dvojice regulátorů, vnitřní typu P a vnější typu PID, byly odladěny na sestaveném modelu v programu Matlab Simulink. Nejprve regulátor P tak, aby docházelo k rychlému sledování žádané úhlové rychlosti. Jeho přenos odpovídá rovnici (18).

$$R_{x2}(p) = k_2 = 0,2 \tag{18}$$

PID regulátor obsahuje trojici proměnných parametrů a jeho ruční ladění je tak složitější. Zároveň kvůli nelinearitě nejdou použít běžné návrhové metody známé z lineárních systémů. Proto byla vnitřní uzavřená smyčka, obsahující nelinearitu, aproximována pro jeden pracovní bod setrvačným článkem druhého řádu [29]. Jejich porovnání při žádané hodnotě úhlové rychlosti 100 °/s je na Obrázku 24. Aproximovaná vnitřní smyčka odpovídá statickému přenosu (19).

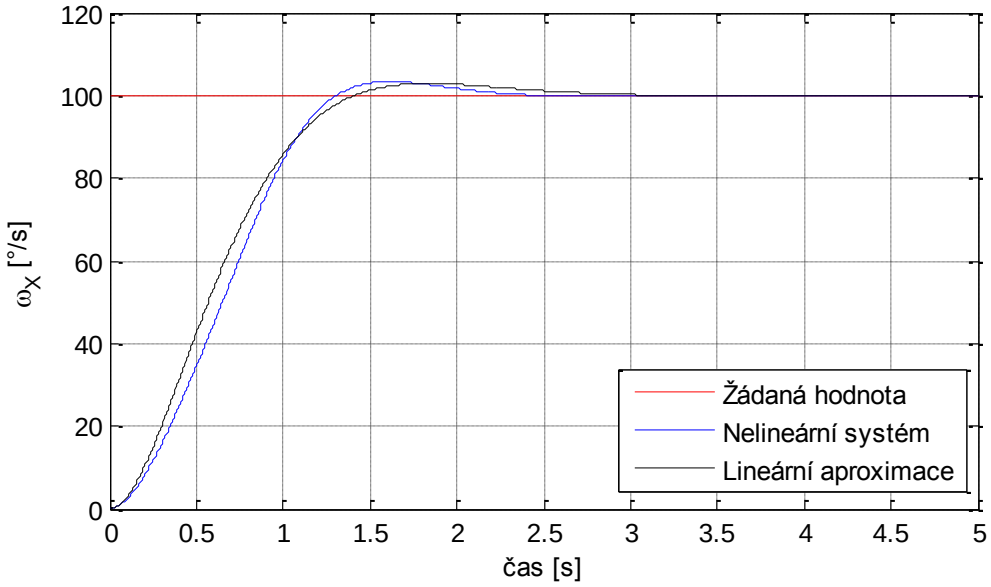
$$F_{w1APR}(p) = \frac{1}{T^2 p^2 + 2T\xi p + 1} = \frac{1}{0,15p^2 + 0,58p + 1} \tag{19}$$

Na získaný lineární systém byla použita návrhová metoda Ziegler-Nicholsova, díky níž byly určeny parametry PID regulátoru vnější smyčky (20).

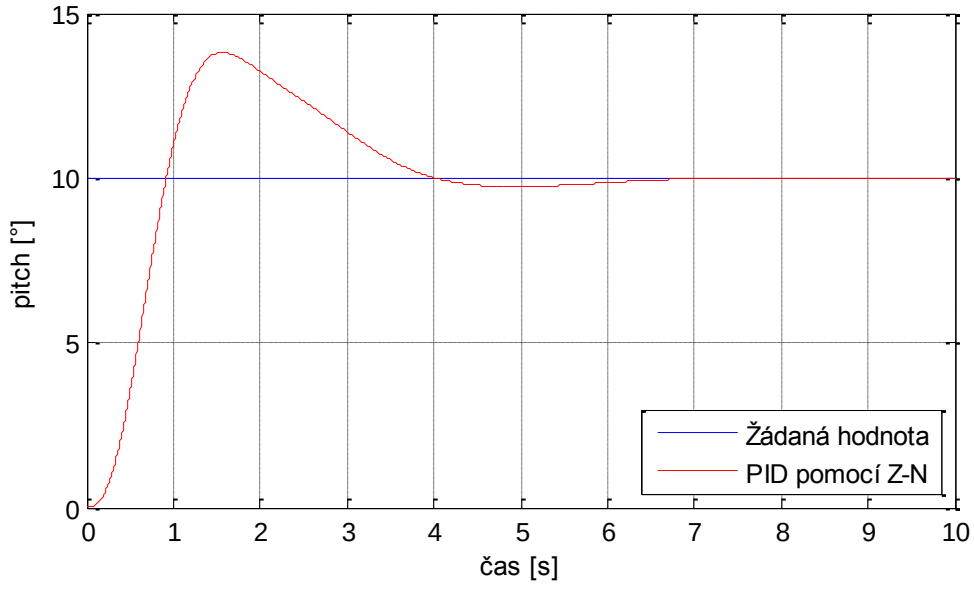
$$R_{X1ZN}(p) = K_R(1 + \frac{1}{T_I p} + T_D p) = 2,292(1 + \frac{1}{1,215p} + 0,304p) \tag{20}$$

Tato metoda je navržena pro optimální vyregulování poruchy a výsledná přechodová charakteristika má kmitavý charakter s překmity o velikosti 10 % - 60 %. Jedná se však

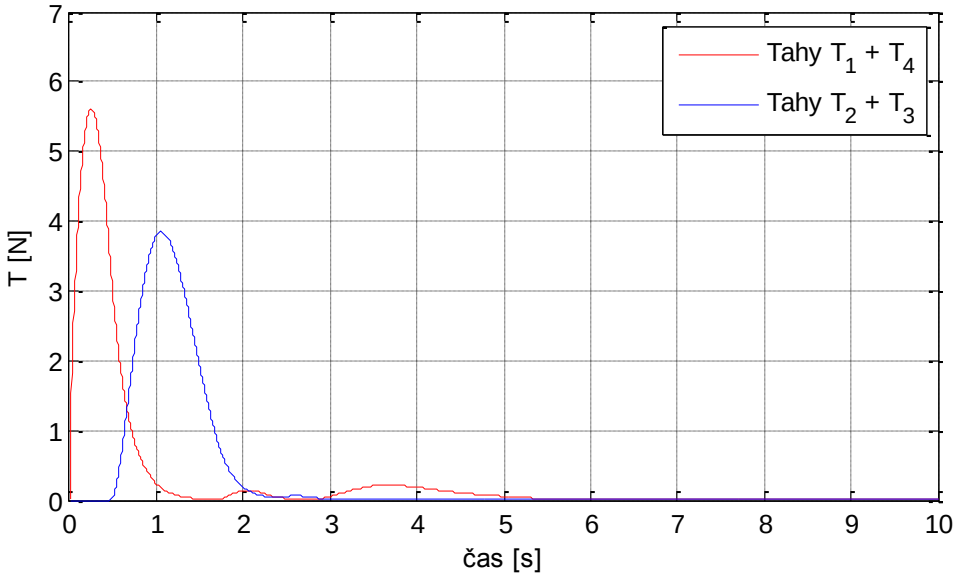
o dobrý počáteční odhad pro další ladění regulátoru, které je provedeno již na reálné soustavě. Výsledek simulace pro změnu sklonu robotu (*pitch*) o 10 ° je na Obrázku 25. Na Obrázku 26 jsou pak průběhy akčních zásahů, které odpovídají dříve uvedenému kinematickému schématu robotu.



Obrázek 24: Porovnání nelineární vnitřní smyčky a její lineární aproximace



Obrázek 25: Přechodová charakteristika vnější smyčky s PID regulátorem podle Z-N



Obrázek 26: Průběhy akčních zásahů ve formě tahů vrtulí

Obdobným způsobem byl sestaven i regulátor pro osu Y, jelikož je chování robotu v obou osách stejné. Liší se mírně pouze momentem setrvačnosti.

6.3 Testování stabilizace

Jelikož je k dispozici konstrukce robotu, funkční hardware a dálkové řízení a také řídicí aplikace pro mikrokontrolér (viz. kapitola Vytvoření FreeRTOS aplikace), může být navržená stabilizace otestována. Regulační struktura (Obrázek 22), obsahující dva regulátory, je realizována v jazyce C tak, aby mohla být implementována do programu.

Žádaná hodnota je nastavována pomocí dálkového řízení, zpětné vazby od výstupů soustavy jsou tvořeny pomocí AHRS jednotky VN-100T. Ta přímo poskytuje hodnoty úhlových natočení a úhlových rychlostí s periodou 5 ms. Se stejnou periodou jsou počítány akční zásahy z regulátorů a po vhodné transformaci jsou aktualizovány otáčky motorů pomocí připojených regulátorů otáček.

Rozvětvený regulační obvod obsahuje P regulátor, který představuje zesílení regulační odchylky a dále regulátor PID. Jeho diskrétní realizace vychází z tvaru uvedeného v (17), kde integrál je nahrazen sumou a derivace je nahrazena diferencí [30]. Tvar PSD regulátoru je uveden v rovnici (21).

$$x_{PSD}(k) = r_0 e(k) + r_i T_{VZ} \sum_{i=1}^{k} e(i) + r_d \frac{(e(k) - e(k-1))}{T_{VZ}} \tag{21}$$

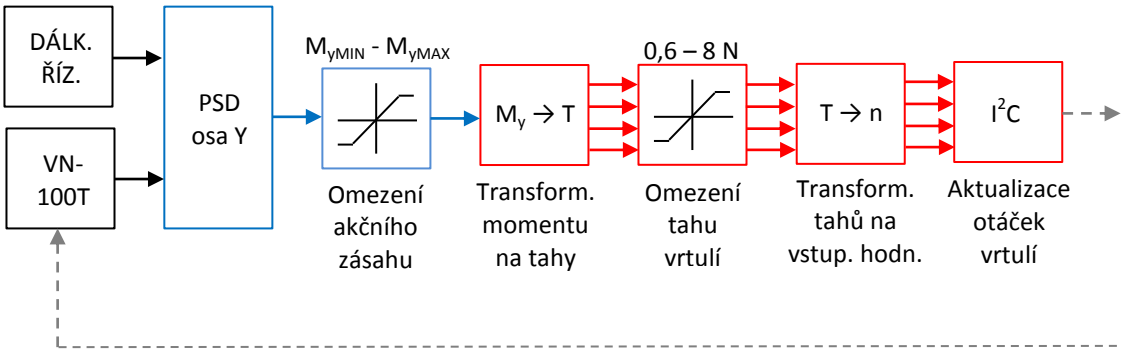
Programová realizace regulátoru pro osu Y je pak následující:

```
w_y = actualOrientation.angSpd[1];
e_y= (actualJoystick.yprtNorm_1_1[1]*30) - actualOrientation.ypr[1];
int_y = int_y + (e_y*Tvz);
if(int_y>8) int_y = 8;
if(int_y<-8) int_y = -8;
der_y = (e_y - e_y_pre)/Tvz;
M_y = ( (Kp_y*e_y + Ki_y*int_y + Kd_y*der_y) - w_y) * Kpw_y; // PID
if(M_y<-4) M_y=-4; if(M_y>4) M_y=4;
e_y_pre = e_y;
```

Výstup regulátoru `M_y` tvoří akční zásah ve formě momentu síly, musí být proto dále převeden na tahy jednotlivých vrtulí. Zápis vychází z inverze transformační matice $\mathbf{R_{TM}}$ (9) a je doplněn o saturaci, která je v akčních členech fyzicky přítomna:

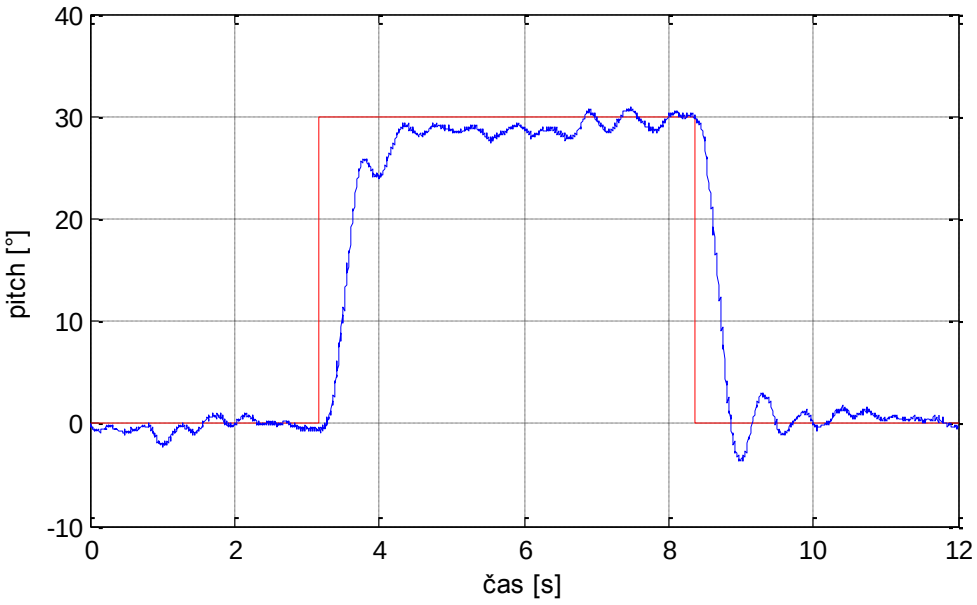
```
T1 = (0.25*T*1)+(-0.9688*M_x)+(+0.9688*M_y)+(+5.0*M_z);
T2 = (0.25*T*1)+(-0.9688*M_x)+(-0.9688*M_y)+(-5.0*M_z);
T3 = (0.25*T*1)+(+0.9688*M_x)+(-0.9688*M_y)+(+5.0*M_z);
T4 = (0.25*T*1)+(+0.9688*M_x)+(+0.9688*M_y)+(-5.0*M_z);

if(T1<0.6) T1=0.6;
if(T2<0.6) T2=0.6;
if(T3<0.6) T3=0.6;
if(T4<0.6) T4=0.6;
if(T1>8) T1=8;
if(T2>8) T2=8;
if(T3>8) T3=8;
if(T4>8) T4=8;
```

Hodnoty tahů jsou dále upraveny tak, aby dopovídaly vstupním hodnotám pro regulátory motorů. Kompletní řetězec úkonů je na Obrázku 27, kde modře jsou označeny úkony realizované pro každou osu rotace zvlášť a červeně společná část řetězce.



Obrázek 27: Řetězec úkonů realizované regulační smyčky

Popsaným způsobem byl realizován výše navržený regulátor pro osu Y a reálný model byl otestován. Byl sestaven testovací přípravek (viz. Příloha 6), který umožňuje robotu vykonávat pouze rotační pohyb v ose Y, což odpovídá jednomu stupni volnosti. Po sérii několika testů byly konstanty regulátory doladěny, ale struktura regulačního obvodu zůstala zachována. Výsledek skokového přechodového děje z nulové výchylky na 30 ° a zpět je na Obrázku 28.



Obrázek 28: Skoková změna řízení na reálném systému

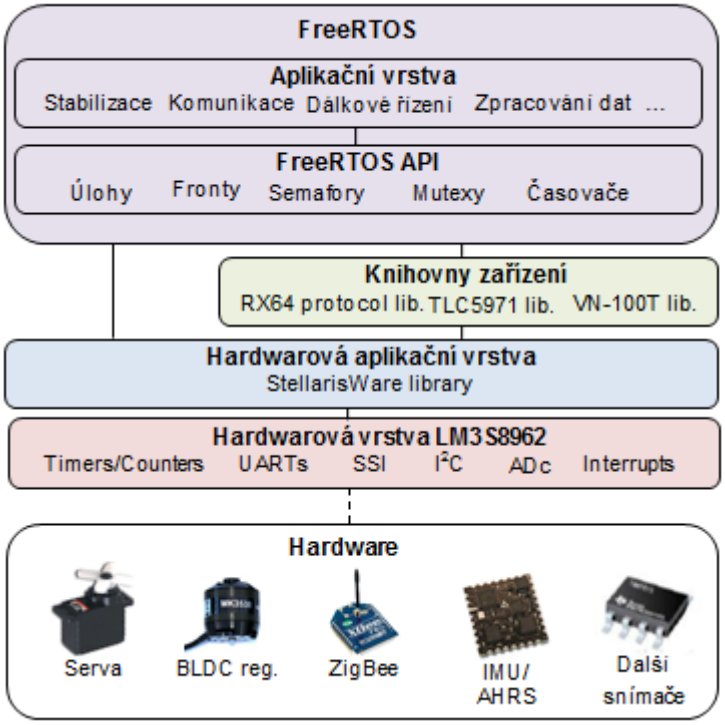
Z průběhu lze vyčíst, že z původního astatického vícerozměrného nelineárního systému, který byl prakticky neřiditelný, se stal díky regulačnímu obvodu systém, který se chová jako statický a dokáže přibližně kopírovat žádanou hodnotu. Matematický model se od reálného testovaného systému liší jistě mnoha aspektech. Například v úchytech robotu na přípravku vzniká díky jeho poměrně vysoké hmotnosti tření, působící jako chyba na vstup soustavy. Další, nelineárně působící chyba, vzniká vlivem podvozku robotu, který je zavěšen pod jeho tělem. Robot má tak tendenci samovolně držet nulový náklon, podobně jako kyvadlo. V blízkosti země jistě vzniká také těžce definovatelná chyba způsobená prouděním vzduchu od vrtulí.

Přesto se dá považovat výsledek testování jako uspokojující. Předmětem dalšího testování bude zajisté snaha vyladit konstanty stávajících regulátorů tak, aby bylo dosaženo optimálnější dynamiky přechodového děje. Předmětem testování může být také vylepšení či konstrukce nového přípravku, který by zanášel do testování menší chybu. Celý proces testování se ukázal náročný také z hlediska prostorových nároků, jelikož se jedná o poměrně velkou konstrukci. Čtveřice rotujících vrtulí a vysoká dynamika robotu vyžaduje dbát i na zajištění bezpečnosti.

7 VYTVOŘENÍ FREERTOS APLIKACE

Operační systém reálného času FreeRTOS již byl představen v kapitole Operační systém FreeRTOS. Tato kapitola se zabývá jeho implementací, konfigurací, tvorbou řídicí aplikace a celkovým programovým řešením pro mikrokontrolér LM3S8962.

Jak bývá zvykem, programové řešení je rozděleno do vrstev. Jeho struktura je znázorněna na Obrázku 29. Hardwarová vrstva mikrokontroléru LM3S8962 představuje rozhraní pro obsluhu připojeného hardwaru. Jednotlivé periferie jsou mapované v paměti a jejich obsluha a práce s daty probíhá prostřednictvím příslušných registrů. Výrobce dodává k řadě ARM Cortex™-M komplexní knihovnu StellarsWare pro práci s periferiemi a veškerými hardwarovými prostředky mikrokontroléru. Tato knihovna tvoří tzv. hardwarovou aplikační vrstvu. Některá připojená zařízení, jako je inerciální jednotka VN-100T, nebo řadič RGB led diod TLC5971, vyžadují složitější programovou obsluhu. Samostatnou vrstvu proto tvoří knihovny funkcí pro práci s nimi, implementující především příslušný komunikační protokol a řídicí příkazy.



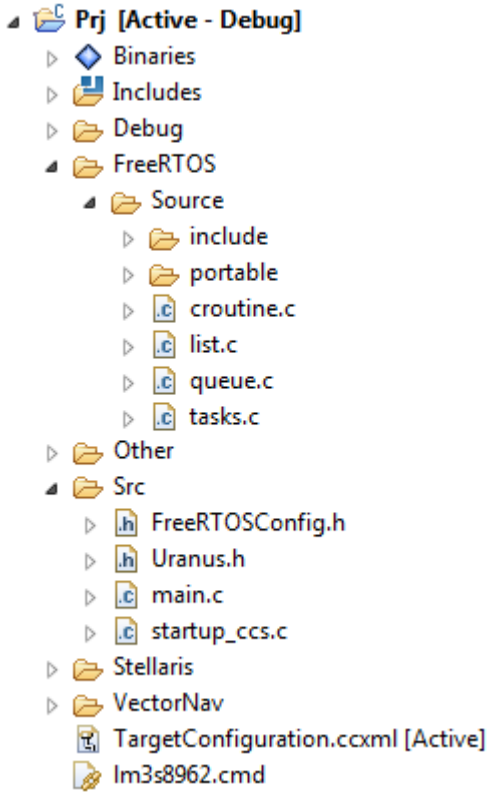
Obrázek 29: Přibližná struktura programového řešení

Vrchní vrstvu programového řešení tvoří OS FreeRTOS. Tato vrstva lze dále rozdělit na jádro systému poskytující API funkce a vrstvu, která představuje samotnou aplikaci.

7.1 Zdrojové soubory

Programové řešení bylo vytvořeno ve vývojovém prostředí Code Composer Studio v4, které je přímo určené pro embedded mikrokontroléry firmy Texas Instruments. CCS je založeno na open-source platformě Eclipse, která je doplněna o řadu nástrojů. Umožňuje vývoj a ladění aplikace, obsahuje editor kódu, správce projektu, kompiler, debugger a simulátor. Integrovaný kompiler podporuje různé architektury mikrokontrolérů (ARM, MSP430) a programovací jazyky C/C++.

Jelikož FreeRTOS vyžaduje specifickou konfiguraci projektu, byl stažen ukázkový projekt nastavený přímo pro mikrokontrolér LM3S8962 [31]. Projekt byl dále doplněn o periferní knihovnu StellarisWare, knihovny zařízení a hlavní funkční a konfigurační soubory. Struktura adresářů a souborů je na Obrázku 30.



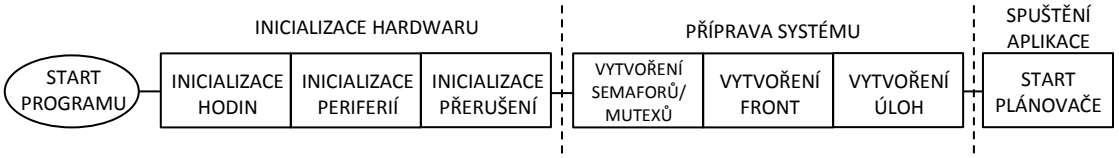
Obrázek 30: Struktura adresářů a souborů projektu

Složka Prj/FreeRTOS obsahuje veškeré zdrojové soubory OS. Je to především trojice `list.c`, `queue.c` a `tasks.c`, což jsou soubory nezávislé na architektuře a dále soubory pro daný mikrokontrolér ve složce Prj/FreeRTOS/Souce/portable. Soubor s hlavním programem `main.c` je ve složce Prj/Src spolu s konfiguračními soubory `FreeRTOSConfig.h` a `Uranus.h`. Soubor `startup_css.c` definuje vektory přerušení. Prj/Stellaris obsahuje aplikační hardwarovou knihovnu a Prj/VectorNav a Prj/Other obsahují knihovny zařízení. Soubory `TargetConfiguration.ccxml` a `lm3s8962.cmd` v hlavním adresáři slouží ke konfiguraci mikrokontroléru. Soubor Prj/Debug/Prj.bin je

přeložený binární soubor určený k naprogramování do flash paměti. Pro sériové programová flash paměti byl použit program LM Flash Programmer.

7.2 FreeRTOS aplikace

Ještě před spuštěním FreeRTOS aplikace musí být mikrokontrolér inicializován. Inicializace se provádí na začátku funkce `main.c` a musí obsahovat konfiguraci hodin mikrokontroléru a veškerých použitých periferií. Pro časování byla zvolena nejvyšší možná frekvence hodin, 50 MHz. Ta je vytvořena z krystalového oscilátoru 8 MHz a PLL frekvenční násobičky. Nesmí být opomenuto nastavení zdrojů přerušení, jejich povolení, nastavení obslužných funkcí a priorit. Po inicializační části jsou vytvořeny semafory, mutexy, fronty a úlohy a následně aktivován plánovač, čímž je spuštěna FreeRTOS aplikace. Posloupnost znázorňuje Obrázek 31.



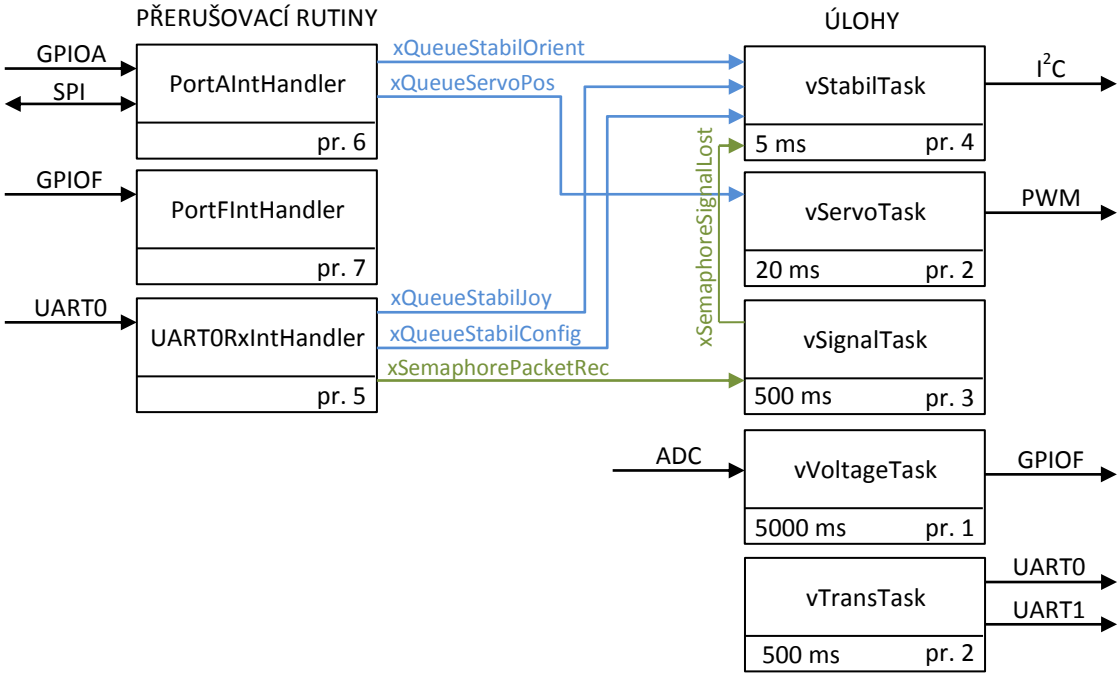
Obrázek 31: Inicializační posloupnost programu

Pro konfiguraci některých parametrů systému slouží hlavičkový soubor `FreeRTOSConfig.h`. Je v něm například nastavena frekvence systémového časovače SysTick, který slouží pro časování úloh. Nastavená frekvence 1000 Hz je vyhovující, jelikož nejrychlejší (stabilizační) úloha pracuje s frekvencí 200 Hz. Dále je zde nastaven plánovač jako preemptivní, počet stupňů softwarových priorit na 5, velikost haldy na 10000 bytů, hardwarová priorita systému a další údaje. Jádro systému má přiřazenou nejnižší hardwarovou prioritu, tj. 7 a API funkce systému mohou využívat přerušení s prioritou nižší než 4 (viz. kapitola Přerušení a priority).

Aplikace je vytvořena s ohledem na maximální využití systémových prostředků. Jednotlivé funkční celky aplikace jsou realizované ve zvláštních úlohách s rozdílným časováním, v případě reakcí na vnější podněty jsou realizovány přerušovacích rutinách (ISR). Pro předávání dat mezi úlohami, případně úlohami a ISR, je využito systémových front. Fronty mají zpravidla jednotkovou délku, aby v případě opožděného vyzvednutí dat nedocházelo ke zpracovávání dat minulých, což v případě real-time řízení nemá význam. Případná synchronizace úloh probíhá pomocí binárních semaforů, čítající semafory použity nebyly. Přístup ke komunikačním kanálům je vždy chráněn pomocí příslušných mutexů, aby nemohlo dojít k vícenásobnému využití stejného hardwarového prostředku více úlohami. Vhodně jsou voleny také softwarové priority úloh a hardwarové priority ISR. Jelikož mají ISR vždy větší prioritu než jádro systému, musí být jejich obsluha co nejrychlejší a případné časově náročnější zpracování je provedeno až v příslušné úloze.

7.2.1 Přehled úloh a ISR

Na Obrázku 32 je přehled všech implementovaných úloh a ISR, jejich propojení pomocí front a semaforů a vyznačení komunikace s okolím. Z obrázku je zřejmý také tok informací, většina jich je z okolního prostředí získávána asynchronně pomocí přerušení od vnějších událostí. Dále jsou předány úlohám, které je po jejich příslušném zpracování odesílají opět ven. U ISR jsou vyznačeny hardwarové priority, u úloh softwarové priority a nastavené časování.



Obrázek 32: Přehled úloh a ISR a jejich vzájemné komunikace

PortAIntHandler – ISR reagující na synchronizační signál z portu A od inerciální jednotky VN-100T. Je to signál pro vyčtení nových dat pomocí SPI. Data jsou předána úloze pro stabilizaci robota a úloze pro stabilizaci kamery.

PrtFIntHandler – ISR reagující na signál z portu F, na jehož jednom pinu je připojen spínač. Spínač je určen pro obecná užití v budoucnu.

UART0RxIntHandler – ISR reagující při zaplnění 8 bytů z 16 příchozího UART0 bufferu. Tato hodnota nastavení je výhodná z toho důvodu, že jeden příchozí paket z Cassandry má velikost právě 8 bytů. Dojde tak k přerušení, až je celý přijat. Následuje rychlé dekódování zprávy dle ID a předání údajů o řízení stabilizační úloze. Zároveň je dán pomocí semaforu signál úloze vSignalTask, která slouží k detekci ztráty signálu.

vStabilTask – úloha zajišťující stabilizaci robota. Má k dispozici informace o orientaci robota a informace o dálkovém řízení. Tyto údaje představují vstupní hodnoty do regulátorů, vypočtené akční zásahy pro jednotlivé motory jsou odesílány po I^2C rozhraní.

Celkově se dá úloha rozdělit na 3 režimy. Po startu aplikace je robot v tzv. *stop mode*, kdy jsou vyčítány data ze snímačů, ale stabilizace a akční zásahy jsou neaktivní. Do *run mode* přejde robot příkazem dálkového řízení a stabilizace a akční zásahy jsou aktivní. V tomto módu probíhá let. Pokud úloha přijme informaci o ztrátě signálu a nachází se v *run mode*, přejde do *auto mode*, kde je prováděna stabilizace robotu, ale bez příkazů dálkového řízení. Příkladem může být automatické přistání.

vServoTask – úloha zajišťující stabilizaci kamery. Pomocí třech PWM výstupů ovládá trojici serv, které natáčí kameru ve třech osách. Při automatickém kamerovém přistávání bude potřeba, aby kamera směrovala stále kolmo dolů na přistávací značku. Úloha má k dispozici informace o orientaci robotu a každých 20 ms aktualizuje hodnotu pro PWM generátory.

vSignalTask – úloha detekující ztrátu signálu. Řídicí data ze systému Cassandra jsou zasílána s proměnnou periodou 50 až 100 ms. Při každém příjmu paketu je dán signál této úloze, která s periodou 500 ms kontroluje, zda signál obdržela. Tato perioda dává jistou toleranci pro ztrátu několika po sobě jdoucích paketů, což může nastat v situaci, kdy se robot nachází na mezi dosahu rádiových modulů. Ztráta signálu může být ovšem vyhodnocena pouze tehdy, když je robot v *run mode*, jindy nejsou periodicky zasílány data.

vVoltageTask – úloha měřící napětí baterie. Pomocí AD převodníku je měřeno napětí baterie, které je přivedeno na vstup převodníku přes napěťový dělič. Kontrola napětí je důležitá z hlediska pádu robotu a nadměrnému vybití baterie. Vybití baterie pod stanovenou úroveň je signalizováno pomocí bzučáku připojeného na port F.

vTransTask – úloha určená pro odesílání dat. Přestože nebyl příjem a zpracování dat v systému Cassandra prozatím implementován, pro ladící účely slouží tato úloha k odesílání různorodých dat, sdílených přes globální proměnné, na UART0, který je připojen na Zig-Bee modul, a UART1, který je přes USB převodník připojen k PC.

Popsaná aplikace, složená z úloh a ISR, realizuje základní funkce robotu. Dálkové řízení, čtení dat z inerciálních snímačů, výpočet stabilizace, akční zásahy a další. Jedná se pouze o jádro, které bude v budoucnu dle potřeb rozšiřováno. Například se počítá s implementací výškové stabilizace pomocí ultrazvukového a tlakového snímače a systémem automatického přistávání, kterým se zabývá práce [6]. Přidáním dalších úloh a jejich propojením pomocí front a semaforů se dá aplikace postupně rozšiřovat.

7.2.2 Optimalizace paměti

V závislosti na velikosti aplikace, počtu běžících úloh a použitých systémových prostředků se liší její nároky na operační paměť. Každé úloze je při jejím vytváření přidělena požadovaná část zásobníku (*stack*), tedy část operační paměti určené pro ukládání stavu registrů a návratové hodnoty při skocích do podprogramů a při obsluze přerušení. Určení potřebné velikosti je složité, proto se tak provádí většinou iterativně. Ze začátku jsou voleny větší hodnoty zásobníku, aby nemohlo dojít k přetečení a

postupně se tato hodnota upravuje, dle jeho využití. Zjistit velikost využitého zásobníku úlohou se dá provést dvěma způsoby. Systémová funkce `uxTaskGetStackHighWaterMark` vrací nejnižší hodnotu zbývajícího zásobníku dané úlohy, které bylo dosaženo během jejího dosavadního běhu. To samé se dá provést pomocí funkce `vTaskList`. Ta vrací nejnižší hodnoty zásobníků u všech úloh, spolu s jejich jménem, prioritou a statusem. Příkladem je následující výpis:

```
TASK            STATUS      PRIOR       STACK       ORDER

IDLE            R           0           64          5
Servo Task      B           2           160         0
Time Task       B           1           174         2
Signal Task     B           3           174         3
Led Task        B           1           176         4
Stabil Task     B           4           397         1
Volt Task       B           1           156         6
```

Z uvedeného výpisu je možné vyčíst, že například úloha `Stabil Task` dosáhla nejnižší úrovně zbývajícího přiděleného zásobníku 397. Při jejím vytváření byla nastavena hodnota 500, což je v tomto případě zhruba pětinásobek využité, a proto může být snížena. Přesto se doporučuje ponechat alespoň dvojnásobek. Hodnoty zásobníku úlohám se zadávají ve slovech (*word*), které má v případě LM3S8962 velikost 4 byty.

Dosud byla řeč o zásobníku, který je přidělen jednotlivým úlohám při jejich vytváření za běhu programu. Ještě před samotnou kompilací programu a nahráním do paměti mikrokontroléru musí být nastavena celková velikost zásobníku. Ten se v Code Composer Studiu nastavuje v nastavení kompilátoru a souboru `lm3s8962.cmd`. Jeho volba není opět jednoznačná, odvíjí se jednak od množství zásobníku, který využijí úlohy systému, ale také od počtu použitých přerušení. Jeho velikost se dá sledovat a ladit pomocí deguggeru.

Druhou věcí, která se dá optimalizovat, je velikost haldy (*heap*). To je část operační paměti využité systémem, se kterou pracují úlohy, fronty a semafory. Například u fronty je potřebná velikost haldy dána jednak její datovou strukturou, ale především počtem a velikostí dat, které bude uchovávat. Velikost celkové haldy se nastavuje taktéž v nastavení kompilátoru a v konfiguračním souboru `FreeRTOSConfig.h`. Její optimalizace je o něco jednodušší, než v případě zásobníku, jelikož se velikost využité části po startu systému již nemění. Funkcí `xPortGetFreeHeapSize` se dá zjistit velikost nevyužité části haldy. O tuto hodnotu může být nastavená velikost haldy snížena.

Postupnou optimalizací byla nastavena velikost zásobníku na 8 kB a velikost haldy na 10 kB. Pomocí debuggeru bylo zjištěno přibližně poloviční využití celkové paměti SRAM. Paměť pro program typu flash má stále velkou rezervu, jelikož aktuální program zabírá 22 kB z celkových 256 kB.

8 ZÁVĚR

Cílem této diplomové práce bylo navrhnout a realizovat řídicí systém pro létající robot typu quadrocopter. Volba vhodného mikrokontroléru vycházela z požadavků systému a zkušeností z předcházejícího vývoje. Byl zvolen mikrokontrolér s jádrem ARM Cortex™-M3, které patří v dnešní době ke špičce v oblasti jednoúčelových aplikací. Integruje 32bitovou architekturu s pokročilou správou přerušení, ochranou paměti a ladícím rozhraním a v případě zvoleného mikrokontroléru LM3S8962 od firmy Texas Instruments je jádro doplněno o 256 kB flash a 64 kB SRAM paměti a množství periferií. Díky taktovací frekvenci až 50 MHz tak poskytuje dostatečný výkon pro realizaci i složitějších stabilizačních algoritmů.

Hardwarové řešení systému dále obsahuje AHRS jednotku VectorNav VN-100T, která pro výpočet orientace využívá konfigurovatelného Kalmanova filtru a umožňuje tak optimálně využít data ze všech inerciálních snímačů. Znalost orientace je využita kromě stabilizace robotu také pro stabilizační jednotku kamery zavěšené pod tělem robotu.

Pro dálkové řízení robotu je využito systému Cassandra-WPF, který je určen pro real-time řízení skupiny průzkumných robotů. Fyzicky je komunikace se systémem zajištěna pomocí dvou Zig-Bee modulů pracujících v pásmu 2,4 GHz. Komunikace probíhá sériově a je implementován jednoduchý protokol na úrovni druhé vrstvy ISO/OSI referenčního modelu. Umožňuje zasílání řídicích pokynů z PC do robotu a také zasílání letových údajů a stavů snímačů v opačném směru.

Jako základ programového řešení byl zvolen operační systém reálného času FreeRTOS. Jedná se o velmi jednoduchý operační systém určený pro mikrokontroléry, díky němuž se výsledné programové řešení stává univerzálnější a přehlednější. Za pomocí dostupných API funkcí, především pro správu úloh a komunikaci mezi nimi, byla vytvořena aplikace integrující základní funkcionalitu robotu. Tou je komunikace s nadřazeným systémem Cassandra-WPF, komunikace se snímači, řízení akčních členů a integrace stabilizačních algoritmů.

Pro otestování hardwarového a softwarového řešení byly navrženy regulátory pro stabilizaci náklonu a sklonu robotu. Jejich návrh vycházel z pochopení chování robotu a sestavení matematického modelu v programu Matlab Simulink. Pro regulaci astatického systému byl navržen regulátor s pomocnou regulovanou veličinou, obsahující regulátor P pro vnitřní a PID pro vnější smyčku. Díky následné implementaci stabilizačního algoritmu do operačního systému mohl být celek otestován na testovacím přípravku. Navržené regulátory umožňují řízení původně velmi nestabilního systému, ovšem před první letovou zkouškou bude potřeba optimalizovat jejich dynamické vlastnosti a provést velké množství dalších testů.

Výsledkem této diplomové práce je odladěné hardwarové a softwarové řešení pro létající robot typu quadrocopter, které tak poskytuje možnost posunout vývoj o krok dál.

Použitý mikrokontrolér a systém FreeRTOS umožňují jednoduchou implementaci dalších algoritmů, nebo připojení nadstavbových modulů či zařízení. Zvláštní pozornost by v budoucnu měla být věnována především stabilizačním algoritmům, jelikož kontrolovatelný let je základním předpokladem pro následující širokou škálu použití. Nabízí se například implementace kamerového systému pro automatické přistávání, kterým se zabývá práce [6], nebo navigačního modulu pro autonomní let za použití GPS navigace.

SEZNAM LITERATURY

[1] GÁBRLÍK, P. *Quadrocopter - stabilizace a regulace*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2010. 75 s. Vedoucí bakalářské práce doc. Ing. Luděk Žalud, Ph.D.

[2] VOMOČIL, J. *Quadrocopter – řídicí jednotka a komunikace*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2010. 64 s. Vedoucí bakalářské práce doc. Ing. Luděk Žalud, Ph.D.

[3] JURAJDA, D. *Quadrocopter – navigační modul*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2011. 54s. Vedoucí bakalářské práce byl doc. Ing. Luděk Žalud, Ph.D.

[4] KŘÍŽ, V. *Řídicí systém vzdušného průzkumného prostředku pro vnitřní prostředí*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2011. 67 s. Vedoucí diplomové práce doc. Ing. Luděk Žalud, Ph.D.

[5] BRADÁČ, F. *Quadrokopter – stabilizace pomocí inerciálních snímačů*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2009. 62s. Vedoucí diplomové práce byl doc. Ing. Luděk Žalud, Ph.D.

[6] VOMOČIL, J. *Systém pro automatické přistávání Quadrocopteru*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2012. 76s. Vedoucí diplomové práce byl doc. Ing. Luděk Žalud, Ph.D.

[7] NAHAS, M. *Implementation of highly-predictable time triggered cooperative scheduler using simple super loop architecture*. International Journal of Electrical and Computer Sciences. 2011, roč. 11, č. 4, s. 33-38. Dostupné z: *http://www.ijens.org/Vol_11_I_04/117404-8989-IJECS-IJENS.pdf*

[8] KUČERA, P. *Operační Systémy Reálného Času – Úvod* [online]. Podklady k předmětu MRTS, FEKT VUT Brno. [cit. 2012-02-15]. Dostupné z: *http://taceo.eu/lectures/mrts/data/01_uvod_cz.pdf*

[9] ZEZULKA, F. *Prostředky průmyslové automatizace*. Brno: VUTIUM, 2004. 146 s.

[10] LI, Q. *Real-Time Concepts for Embedded Systems*. 1. vyd. San Francisco: CMP, 2003. ISBN 1-57820-124-1.

[11] *FreeRTOS - A Free professional grade RTOS*: *The FreeRTOS Project* [online]. 2012 [cit. 2012-05-21]. Dostupné z: *http://http://www.freertos.org/*

[12] *Free and open ARM Cortex M3 and Cortex M0 RTOS*: *CooCox CoOS* [online]. 2011 [cit. 2012-05-21]. Dostupné z: *http://www.coocox.org/CoOS.htm*

[13] BARRY, R. *Using the FreeRTOS™ Real Time Kernel*: *A Practical Guide*. 1. vyd. 2010. ISBN 9781446169148.

[14] Processors - ARM: Processors. © ARM LTD. *ARM - The Architecture For The Digital World* [online]. 2012 [cit. 2012-05-21]. Dostupné z: *http://www.arm.com/products/processors/index.php*

[15] TEXAS INSTRUMENTS INCORPORATED. *Stellaris® LM3S8962 Microcontroller*: *DATA SHEET* [online]. Revize 11108. 2011 [cit. 2012-05-21]. Dostupné z: *http://www.ti.com/lit/ds/symlink/lm3s8962.pdf*

[16] Forums - Stellaris® ARM® Cortex™-M Microcontroller - E2E. TEXAS INSTRUMENTS INCORPORATED. Analog, Embedded Processing, Semiconductor Company, Texas Instruments [online]. 2012 [cit. 2012-05-21]. Dostupné z: *http://e2e.ti.com/support/microcontrollers/stellaris_arm_cortex-m3_microcontroller/f/default.aspx*

[17] VECTORNAV TECHNOLOGIES. *VectorNav: Orientation Sensors - IMU - AHRS* [online]. 2012 [cit. 2012-05-21]. Dostupné z: *http://www.vectornav.com/*

[18] VECTORNAV TECHNOLOGIES. *VectorNav: VN-100 User Manual* [online]. 2009 [cit. 2012-05-08]. Dostupné z: *http://www.vectornav.com/Downloads/Support/UM001.pdf*

[19] VECTORNAV TECHNOLOGIES. *Tuning the Performance of the VN-100*: *Application Note 001* [online]. [cit. 2012-05-21]. Dostupné z: *http://www.vectornav.com/Downloads/Support/AN001.pdf*

[20] VECTORNAV TECHNOLOGIES. *Filter Dynamic Disturbance Tuning*: *Application Note 005* [online]. [cit. 2012-05-21]. Dostupné z: *http://www.vectornav.com/Downloads/Support/AN005.pdf*

[21] TEXAS INSTRUMENTS. TLC5971: *12-Channel, 16-Bit, Enhanced Spectrum, PWM, RGB, LED Driver with 3.3-V Linear Regulator* [online]. SBVS146A. 2010 [cit. 2012-05-21]. Dostupné z: *http://www.vectornav.com/Downloads/Support/AN005.pdf*

[22] ROBOTIS. *Dynamixel RX-64*: *User's Manual* [online]. 2006 [cit. 2012-05-21]. Dostupné z: *http://www.megarobot.net/cj/manualy/robotis/RX64_aj.pdf*

[23] OSI model. In: *Wikipedia*: *the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001-2012 [cit. 2012-05-21]. Dostupné z: *http://en.wikipedia.org/wiki/OSI_model*

[24] ŽALUD, L.; FLORIÁN, T.; KOPEČNÝ, L.; BURIAN, F. *CASSANDRA - Heterogeneous Reconnaissance Robotic System for Dangerous Environments*. In Proceedings of 2011 IEEE/ SICE International Symposium on System Integration. Kyoto: Kyoto University, 2011. s. 1-6. ISBN: 9781457715228.

[25] ŠOLC, F. *Modelling and Control of a Quadrocopter*. Advances in Military Technology, 2010, roč. 5, č. 2, s. 29-38. ISSN: 1802- 2308.

[26] BOUABDALLAH, S. *Design and control of quadrotors with application to autonomous flying*. Lausanne, 2007. Dostupné z: *http://biblion.epfl.ch/EPFL/theses/2007/3727/EPFL_TH3727.pdf*. Thèse sciences no 3727. Ecole polytechnique fédérale de Lausanne.

[27] HALLIDAY, D., R. RESNICK a J. WALKER. *Fyzika*: *Část 1, Mechanika*. Brno: VUTIUM, PROMETHEUS, 2000.

[28] BLAHA, P. a P. VAVŘÍN. *Řízení a regulace* I: *Základy regulace lineárních systémů - spojité a diskrétní*. VUT Brno, 2005.

[29] ŠOLC, F., P. VÁCLAVEK a P. VAVŘÍN. *Řízení a regulace II*: *Analýza a řízení nelineárních systémů*. v. 1.16. VUT Brno, 2009.

[30] PIVOŇKA, P. *Číslicová řídicí technika*. VUT Brno, 2003.

[31] FreeRTOS porting on Code Composer Studio 4 (LM3Sxxxx processor). *EMBEDDED'S HAVEN*: *Embedded Linux, RTOS and Parallel Programming* [online]. 2012 [cit. 2012-05-21]. Dostupné z: *http://www.embeddedheaven.com/freertos-code-composer-studio-4-lm3sxxxx.htm*

SEZNAM ZKRATEK

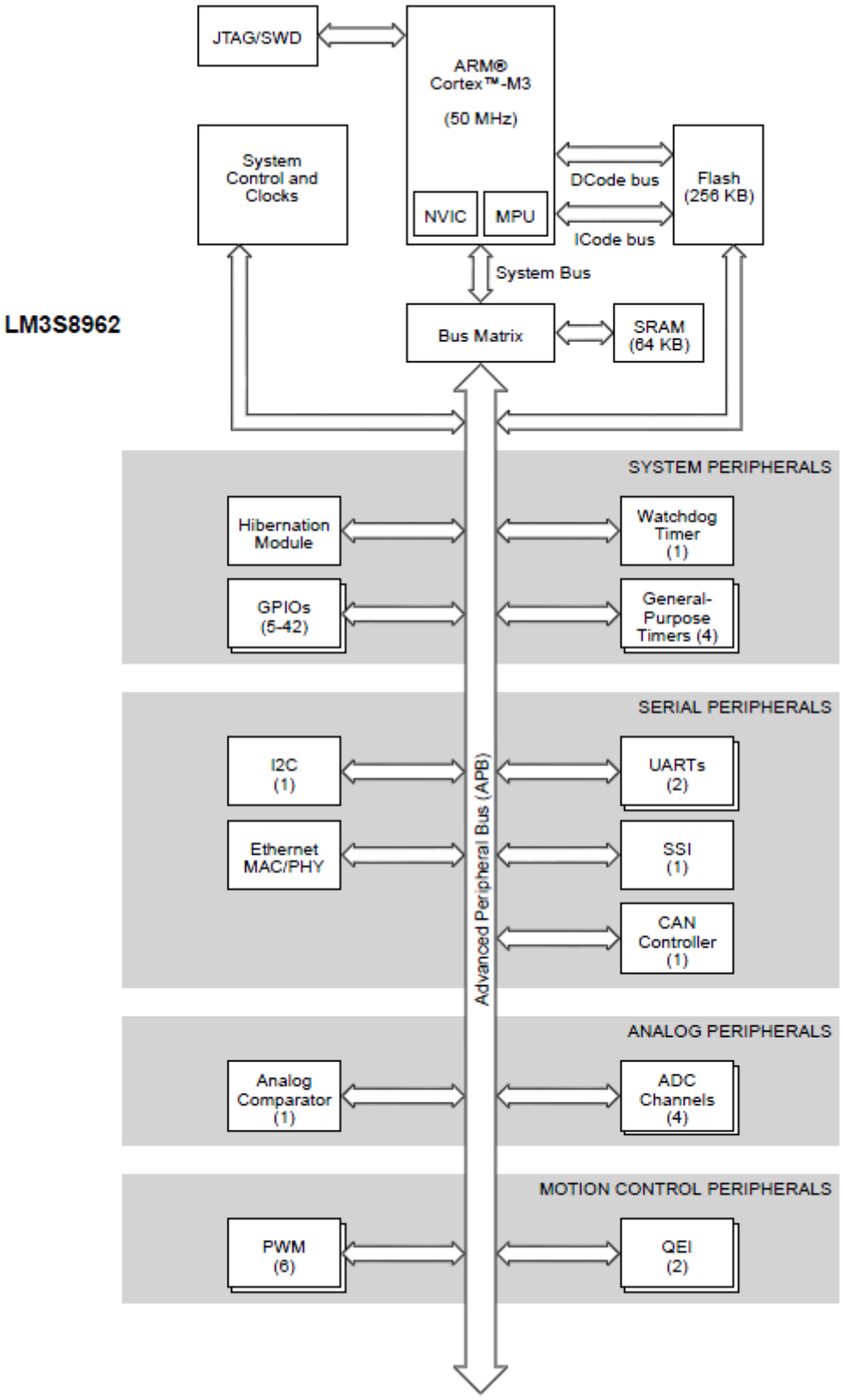
AD	Analogově-digitální
AHRS	Orientační snímač (*Attitude and Heading Reference System*)
API	Rozhraní pro programování aplikací (*Application Programming Interface*)
BLDC	Bezkartáčový stejnosměrný motor (*Brushless DC motor*)
CAN	Komunikační rozhraní *Controller Area Network*
FIFO	Fronta typu *First In, First Out*
GPIO	I/O piny pro obecné použití (*General-Purpose Input/Outputs*)
GPS	Globální družicový polohový systém (*Global Positioning System*)
HID	Rozhraní člověk-PC (*Human Interface Device*)
I^2C	Sériová sběrnice *Inter-Integrated Circuit*
ISR	Rutina obsluhy přerušení (*Interrupt Service Routine*)
JTAG	Standart pro testování a ladění (*Joint Test Action Group*)
KF	Kalmanův filtr
IMU	Inerciální jednotka (*Inertial Measurement Unit*)
MPU	Jednotka pro ochranu paměti (*Memory Protection Unit*)
NMI	Nemaskovatelné přerušení (*Non-Maskable Interrupt*)
NVIC	Jednotka pro správu přerušení (*Nested Vectored Interrupt Controller*)
OS	Operační systém (*Operating System*)
PC	Osobní počítač (*Personal Computer*)
PWM	Pulsně šířková modulace (*Pulse Width Modulator*)
QEI	Dvoukanálový inkrementální enkodér (*Quadrature Encoder Interface*)
RTOS	Operační systém reálného času (*Real-Time Operatin System*)
SSI	Synchronní sériové rozhraní (*Synchronous Serial Interface*)
TI	Texas Instruments
UART	Univerzální asynchronní sériové rozhraní (*Universal Asynchronous Receiver Transmitter*)
UAV	Bezpilotní létající stroj (*Unmanned Aerial Vehicle*)
USB	Univerzální sériové rozhraní (*Universal Seriál Bus*)

SEZNAM SYMBOLŮ

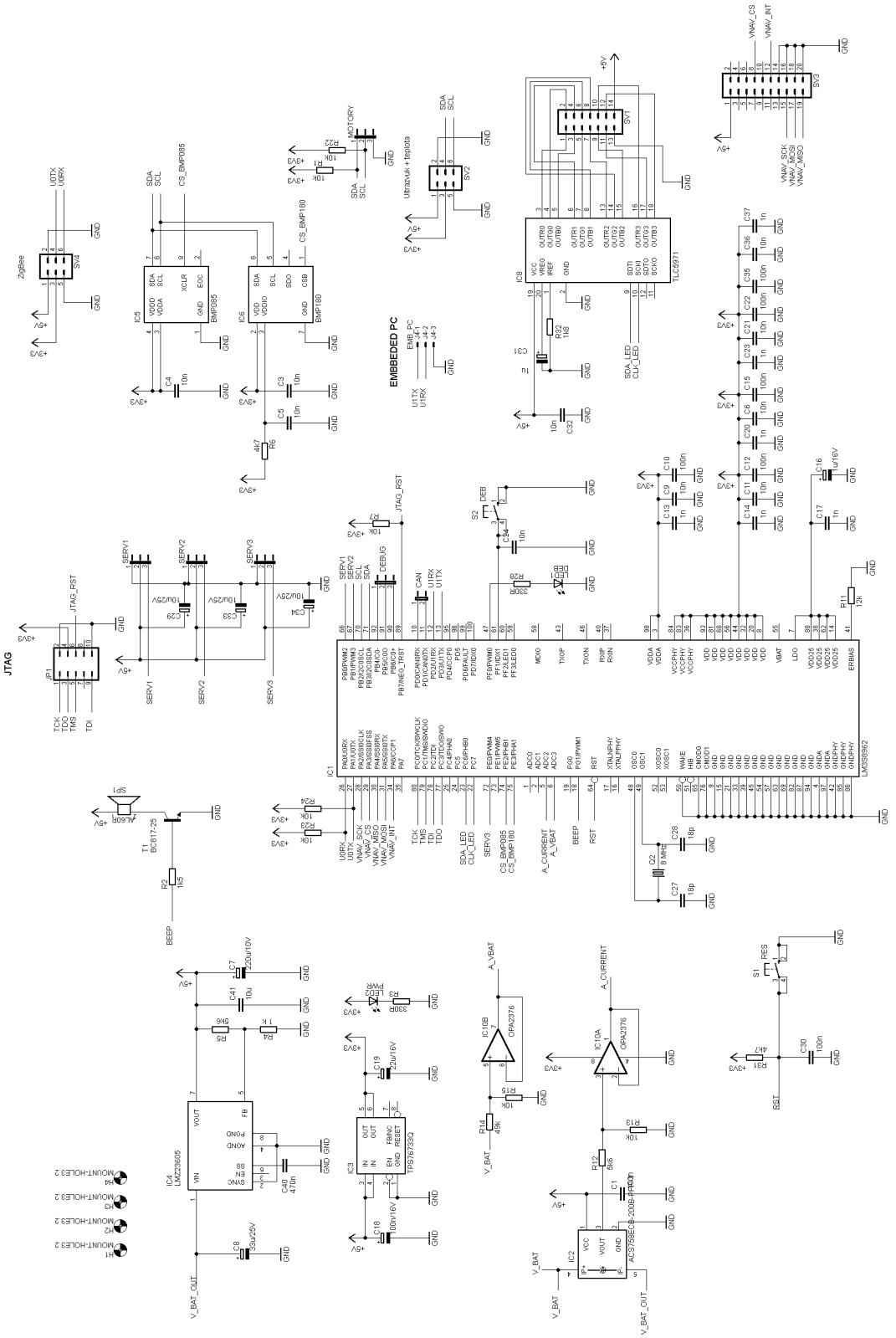
J_X, J_Y, J_Z	momenty setrvačností v osách robotu
l	délka ramene robotu
m	hmotnost robotu
$\mathbf{M_X}$, $\mathbf{M_Y}$, $\mathbf{M_Z}$	momenty sil v osách robotu
$\mathbf{M_{R1}}$, $\mathbf{M_{R2}}$, $\mathbf{M_{R3}}$, $\mathbf{M_{R4}}$	reakční momenty vrtulí
$\mathbf{T_1}$, $\mathbf{T_2}$, $\mathbf{T_3}$, $\mathbf{T_4}$	tahové síly vrtulí
$\mathbf{X_E}$, $\mathbf{Y_E}$, $\mathbf{Z_E}$	osy souřadného systému Země
$\mathbf{X_U}$, $\mathbf{Y_U}$, $\mathbf{Z_U}$	osy souřadného systému robotu
φ, θ, ψ	úhlová natočení (*roll*, *pitch*, *yaw*)
ε_x, ε_y, ε_z	úhlová zrychlení
ω_x, ω_y, ω_z	úhlové rychlosti
ω_1, ω_2, ω_3, ω_4	rychlosti otáčení vrtulí

SEZNAM PŘÍLOH

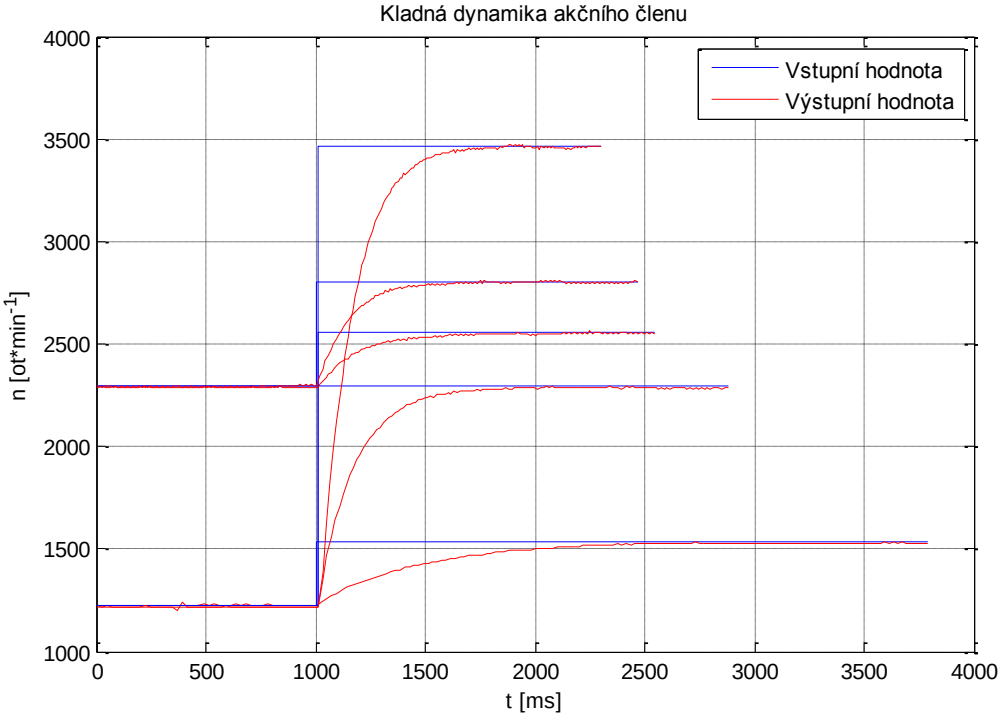
Příloha 1. High-Level blokový diagram mikrokontroléru LM3S8962

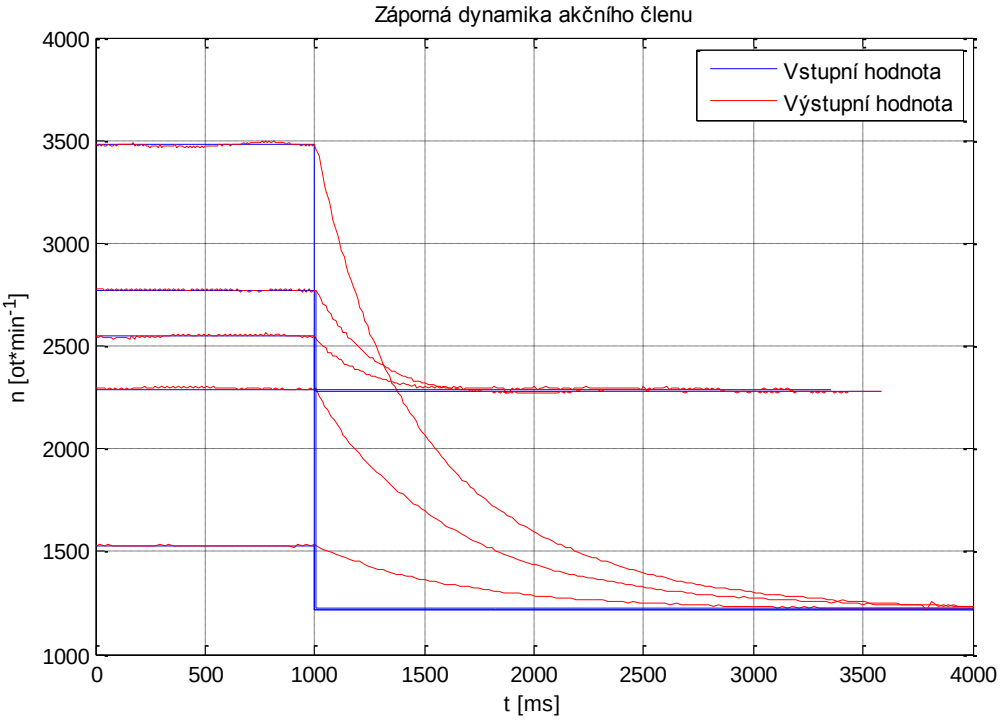


Příloha 2. Schéma zapojení řídicí desky

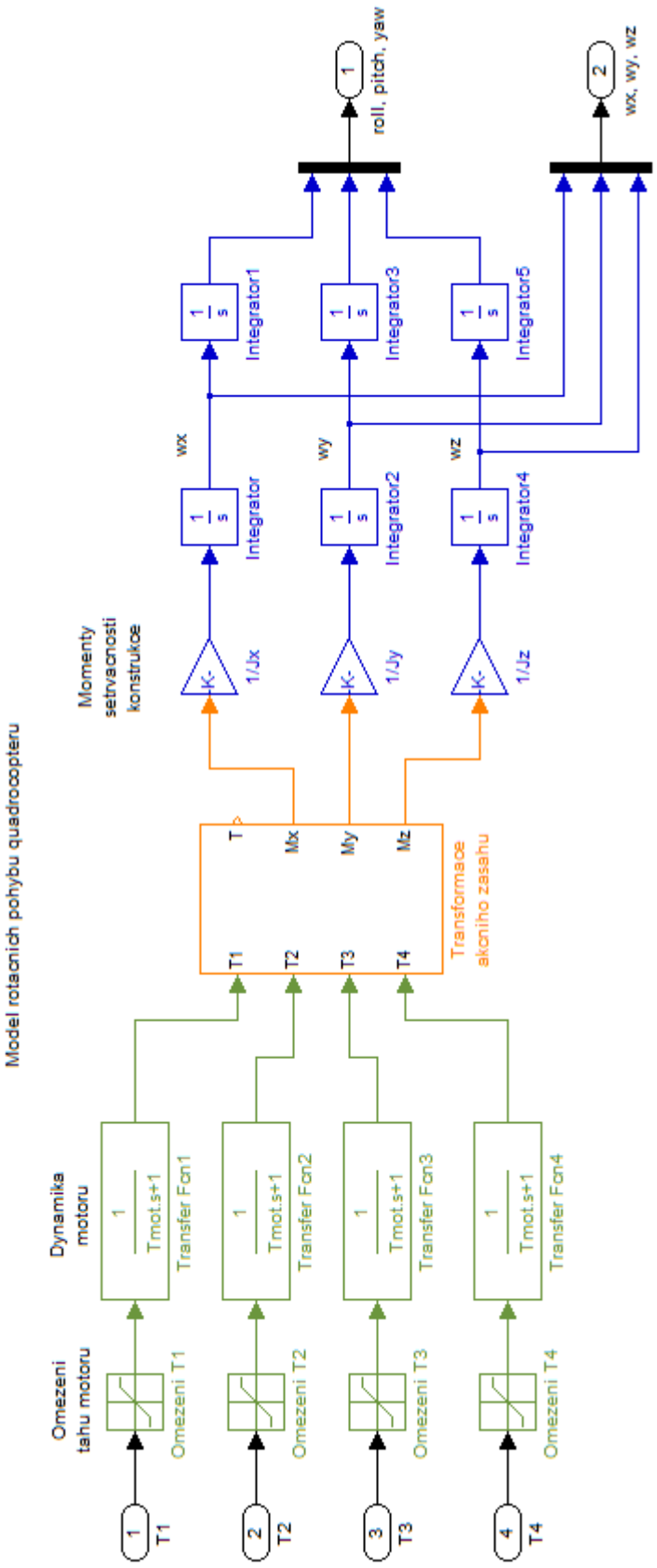


Příloha 3. Identifikace akčního členu

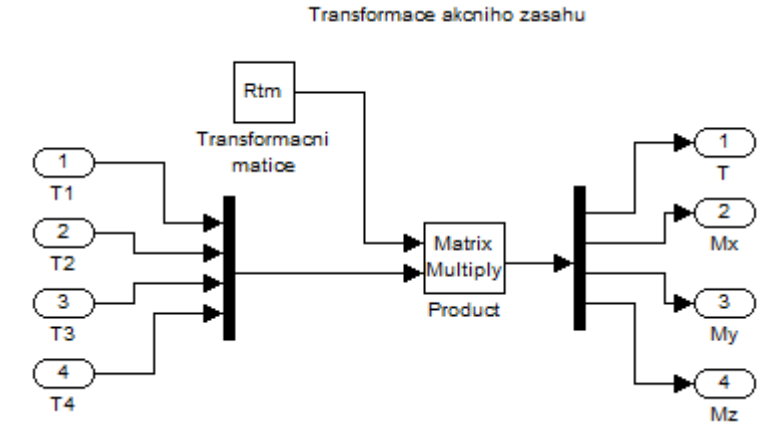


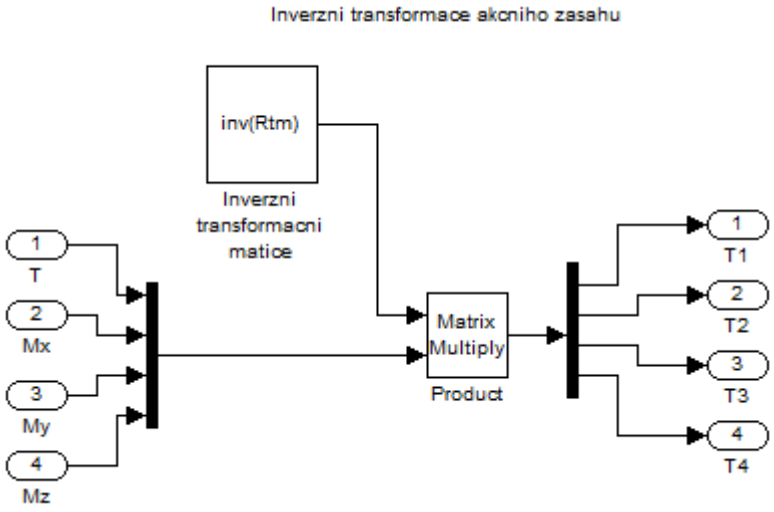


Příloha 4. Model quadrocopteru v programu Matlab Simulink

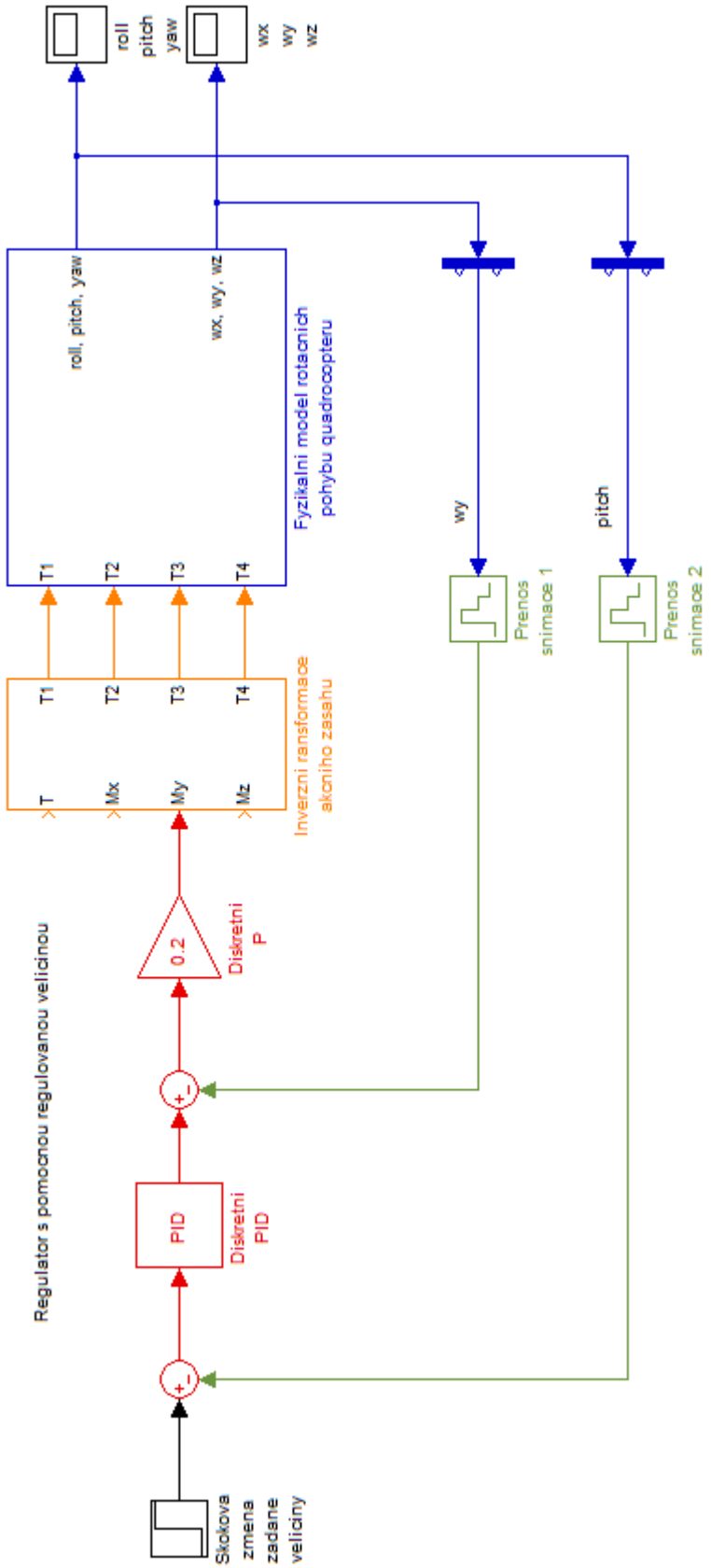


Příloha 4. Model quadrocopteru v programu Matlab Simulink





Příloha 5. Regulační schéma pro osu Y v programu Matlab Simulink



Příloha 6. Fotodokumentace