MRTS – Systémy reálného času

Ovládání robotického manipulátoru ROB 2-6

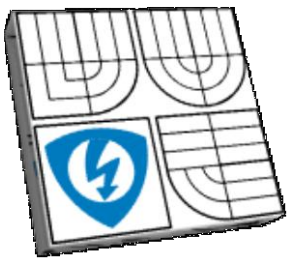
Půlsemestrální projekt č.2

Zadání:

- Vytvořte interpretační jazyk pro ovládání poloh jednotlivých os manipulátoru ROB 2-6 připojeného ke kartě PIO-821. V tomto jazyce vytvořte různé sekvence příkazů pro manipulátor a uložte je do souborů. V RTX a WIN32 naprogramujte aplikaci, která dle požadavků uživatele otevře konkrétní soubor a danou sekvenci příkazů vykoná.

Pravidla

1. Jeden nebo dva studenti pracují na jednom projektu dle zadání výše.
2. Projekt se obhajuje v zápočtovém týdnu, tj. **16.12.2013** v SD2.92; přijďte do cvičení, do kterého patříte. Student/tým si k obhajobě připraví krátkou **prezentaci** a projekt názorně demonstruje. Projekt musí v laboratoři nebo na přineseném počítači fungovat.
3. Do **13.12.2013** mi zašlete zdrojové kódy vašich aplikací a dokumentaci (viz níže). Zdrojové kódy pro RTX aplikace musí být vytvořeny ve vývojovém prostředí Visual Studio .NET 2005 (nebo vyšším) v jazyce C nebo C++. Zdrojové kódy pro WIN32 aplikace mohou být vytvořeny i v jiném vývojovém prostředí či programovacím jazyce. Projekt zazipujte a zašlete e-mailem. Do předmětu mailu vložte klíčové slovo MRTS. Z projektu před zazipováním odstraňte případné adresáře Release, Debug, RTSSRelease, RTSSDebug, aby zip soubor neobsahoval žádné obj, lib, dll, exe, rtss... a jiné exekuční části.
4. Součástí projektu je také **stručná dokumentace**, tj. co to umí, jak to funguje, jak se to ovládá**. Žádné teoretické rozbory** či závěry **nejsou přípustné**; berte to jako **manuál k produktu**. Výpisy kódu uveďte pouze tehdy, když vymyslíte nějakou geniální konstrukci. Doporučený rozsah dokumentace: **1 až 4 strany A4** včetně obrázků, grafů a tabulek. Dokumentaci vytvořte ve statickém HTML formátu, bez frames a ccs nebo **ve formátu pdf**. Dokumentace může sloužit i jako prezentace během obhajoby.
5. Projekt musí využívat **asynchronní verzi funkce LogMessage(...)**, viz 9. úkol cvičení .



Vedoucí: Kučera P.
Vypracoval: Davídek D.; Sliž J.

2013@VUT FEKT

Úvod

Cílem tohoto půl-semestrálního projektu je vytvořit interpretační jazyk pro manipulátor ROB 2-6. Program po přečtení textového souboru se sekvencí instrukcí, bude nastavovat robotický manipulátor postupně do poloh dle pokynů v tomto souboru.

Vypracování

V projektu neřešíme úlohu inverzní kinematiky, pouze nastavujeme pomocí pulzů natočení jednotlivých servomechanizmů. Pro servomechanizmy 1-3, je vhodné, aby při přesunu na novou pozici byla tato pozice nastavována postupně pomocí signálu tvaru rampy, aby nedošlo k příliš velkým rychlostem celé konstrukce a vytvoření nebezpečného pracovního prostředí. Proto je implementována funkce non-fixedPositioning (ovládání bez zpětné vazby).

Celkem 2 projekty VS2010:

1. roboArm – RTSS = ovládá ruku a načítá soubor daný vstupním param.
2. roboArm_starter – WIN32 = spouští roboArm RTSS a předává mu parametr.

Multitask

Implementována jsou celkem 2 vlákna

1. thread – pro neblokující logovací funkci
2. thread – pro zapis do Digital Output registru
3. thread – není implementován pro měření zpětné vazby

Log msg thread

Thread obstarávající asynchronní logování skrze mutex.

PWM_Thread -

V jednom vlákně běží proces, který kontroluje nastavení střídy jednotlivých serv uložených v jednotlivých instancích třídy `C_servoMotor`.

Nazačátku hlavní vnitřní PWMtic smyčky, se nastaví bity všech serv do nuly a pote se ve foru opakovaně čeká po intervalu dostatečně malém oproti časovým konstantám serv = 100us. Inkrementuje se proměná PWMtic a při překročení meze se buď pouze vynuluje; nebo se vyskočí z cyklu PWMticů, načte se další fáze (další prostorové uspořádání = úhly jednotlivých serv), podle toho jestli se překročila mez pro novou periodu PWMperiod_interval[100Hz] nebo pro délku fáze PWMphase_interval[dle control souboru].

Phases

V standartním kontejneru typu list jsou uloženy instance třídy C_spatialConfiguration jenž obsahují informace o nastavení serv v jednotlivých fázích, tj. natočení (resp. intervalOne = jak dlouho drží jedničku), jestli mají najíždět po celou dobu trvání fáze (rampa = not fixedPositioning) délku fáze a další.

Tyto informace jsou na počátku načteny z přeparsovaného control-souboru, popřípadě je za ně dle definovaného makra přidána testovací sekvence fází. Před všemi fázemi je vložena iniciální fáze kdy všechny serva jsou v délce makrem definovaného intervalu nastaveny rampou do výchozích pozic (90°).

Protože ve vláknu se na začátku zapisují nuly a čeká se, dokud se nemá zapsat jednička

→intervalZero je čas v jedné periodě PWM, který je daný bit serva v nule a intervalOne (= PWM_period – intervalZero) je čas v jedničce.

Thread ADC

Pro nedostatek času není dopsán. Měl by přes mutex zapisovat přes mutex do dvojitého bufferu měření pro všechny 3 Feedbacková serva. Z tohoto bufferu si přes mutex vyčítá threadPWM hodnotu vždy po nové PWM periodě a koriguje touto naměřenou hodnotou, nastavovanou hodnotu intervalOne daného serva.

Servomechanizmy

Rozsah serv min-max = 500 – 2500 us

2000 us / 180deg
10 us / 1deg

Popis souborů

LogMessage.h/cpp

Slouží k asynchronnímu logování. Obsahuje třídu `C_CircBuffer`, která obsahuje kruhový buffer. Další třídou je `C_LogMessageA`, která obsahuje zmíněný buffer a metody pro zápis do bufferu a zápis bufferu do souboru ošetřené mutexem. K dispozici jsou také metody pro spuštění a zastavení logování.

Třída `C_CircBuffer`

- Třída tvoří kruhový buffer s využitím pole o definované velikosti.

Metody

Private:
- `char ReadOne();`
  - Zápis jednoho znaku do bufferu.
- `void WriteOne(char in);`
  - Čtení jednoho znaku z bufferu.

Public:
- `C_CircBuffer();`
  - Konstruktor.
- `~C_CircBuffer();`
  - Destruktor.
- `unsigned int Write(char *in);`
  - Zápis textového řetězce do bufferu.
- `unsigned int Read(char* out);`
  - Čtení textového řetězce z bufferu.
- `bool IsEmpthy();`
  - Indikace prázdného bufferu.

Třída `C_LogMessageA`

- Třída je určená k asynchronnímu logování. Obsahuje kruhový buffer, do kterého je možné vložit zprávu a následně zapsat zprávu do logovacího souboru. Kritická sekce je ošetřena mutexem.
- Pro logování z kteréhokoli místa programu se předpokládá **globální instance třídy**.
- Vyžaduje **vytvoření vlákna**, ve kterém se volá metoda pro zápis do souboru:

```
while(logMsg->GetState())
    {
            logMsg->WriteBuffToFile();
            Sleep(10);
    }
```

  Před vytvořením vlákna je třeba spustit logování metodou `LoggingStart()`. Pro ukončení logování a tedy i vlákna k tomu určenému, se volá metoda `LoggingStop()`.
- Pokud je logování spuštěno, jak je popsáno výše, tak je možné zapsat zprávu metodou `PushMessage(…)`.

Metody

Public:
- `C_LogMessageA();`
  - Konstruktor.
- `~C_LogMessageA();`
  - Destruktor.
- `unsigned int PushMessage(char* in, int iSeverity);`
  - Vložení zprávy do bufferu.
  - Vstupní parametry:
    - `in` – textový řetězec
    - `iSeverity` – priorita zprávy (meze definované v .h)
  - Vrací: 0 – úspěch, 1 – logování nebylo spuštěno, zprávu jen vypíše do konzole
- `unsigned int WriteBuffToFile();`
  - Zápis bufferu do logovacího souboru. Volá se v samostatném vlákně, popis výše.
  - Vrací: 0 – úspěch
- `void LoggingStart();`
- `void LoggingStop();`
- `bool GetState();`

C_servoMotor.h/cpp

Soubor třídy jednotlivých serv. Bližší popis je v .cpp zdrojového kódu.

C_roboticManipulator.h/cpp

Soubor třídy celého manipulátoru, obsahuje pole 6ti serv typu C_servoMotor a std::list<C_spatialConfiguration> phase pro ukládání jednotlivých fází. Další popis je v .cpp zdrojového kódu.

C_spatialConfiguration.h/cpp

Soubor třídy jednotlivých fází. Bližší popis je v .cpp zdrojového kódu.

Ostatní jsou uvedeny ve zdrojovém kódu.

Čtení ze souboru

Na počátku se přečte celý soubor do globálního dynamicky alokovaného pole charu.
Poté je parser postupně rozseká a vloží získané informace do listu fází.

Syntaxe ovládacího souboru

Soubor je typu txt ASCII. Parser reaguje na příkazové znaky W=<>. Počet znaků úhlu je vždy čtyři (nula stupňů pro 3tí servo je tedy 2=0000)

Příklad řádků:

příkaz	**popis**	**Příklad uspořádání příkazu v souboru**

nastav uhel serva
tzn. fixedPositioning
[=]
	i_servo[0-5]=uhel[0000-1800]{středník}
	i_servo = index serva
	uhel = nastavit hned - od 0 do 180° po desetinách
	středník = ukončovací znak
		i=xxxx; i=xxxx ;i=xxxx;
		i=xxxx;
		i=xxxx;
		i=xxxx;

Přejdi k úhlu s lineárním přírůstkem v čase na daném časovém intervalu fáze **[<] [>]**	i_servo[0-5]>uhel[0000-1800]{středník} **i_servo** > index serva **uhel** = konečný „po rampě“ - od 0 do 180° po desetinách **středník** = ukončovací znak	i>xxxx; i>xxxx; i>xxxx; i>xxxx; i>xxxx; i>xxxx;
čekej s nastavenými úhly = přidej tuhle fázi **Další data odpovídají další fázi** **[W]**	wait[W]time[yyyyyy*1ms]{středník} **wait** = příkazový znak **time** = čas v násobcích 1ms – maximální je uvedena v roboArm.h **středník** = ukončovací znak	i=xxxx; i=xxxx; i=xxxx; i=xxxx; i=xxxx; i=xxxx; Wyyyyyyy;

Příklad souboru:

Nastav všechny serva do půlky (90°) na 5[s] = 5000[ms] a potom během 5ti sekund lineárně (bez zpětné vazby) přejdi na 0 stupňů (všemi servy)

```
0=0900;1=0900;2=0900;3=0900;4=0900;5=0900;
W50000;
0>0000;1>0000;2>0000;3>0000;4>0000;5>0000;
W50000;
```

Použitá literatura

[1] Zadání projektu http://taceo.eu/mrts_zadani_projektu.php
[2] C++ list reference http://www.cplusplus.com/reference/list/list/