ZÁPADOČESKÁ UNIVERZITA V PLZNI
FAKULTA ELEKTROTECHNICKÁ

KATEDRA APLIKOVANÉ ELEKTRONIKY A TELEKOMUNIKACÍ

BAKALÁŘSKÁ PRÁCE

Tvorba software pro ovládání robotické ruky

Michal Horáček **2015**

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta elektrotechnická
Akademický rok: **2014/2015**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Michal HORÁČEK**
Osobní číslo:	**E12B0261P**
Studijní program:	**B2612 Elektrotechnika a informatika**
Studijní obor:	**Elektronika a telekomunikace**
Název tématu:	**Tvorba software pro ovládání robotické ruky**
Zadávající katedra:	**Katedra aplikované elektroniky a telekomunikací**

Zásady pro vypracování:

1. Proveďte rešerši existujících vhodných přístupů k řízení robotických manipulátorů.
2. Navrhněte vhodnou aplikaci pro použití softwarového ovládání robotických manipulátorů.
3. Zhodnoťte a navrhněte vhodné hardwarové a softwarové platformy pro vývoj aplikace na ovládání robotické ruky.
4. Vytvořte aplikaci pro ovládání robotické ruky s možností předepsání jejího pohybu a otestujte ji na několika vhodných příkladech.

Abstrakt

Tato práce se zabývá problémem návrhu aplikace pro ovládání robotického manipulátoru v jazyce C++. V úvodní části jsou shrnuta současná hardwarová i softwarová řešení dostupná na trhu. V další části je řešena volba vhodné platformy a propojení jednotlivých prvků systému, tedy komunikace mezi ovládací deskou a PC. Následně je popsán algoritmus samotného řízení robotické ruky a jeho zařazení do ovládací knihovny. Dále práce také zahrnuje návrh grafického rozhraní ovládací aplikace a detaily jeho implementace s použitím Qt frameworku. V poslední části je detailně popsán způsob ovládání robotické ruky z pohledu uživatele a možné vstupy a výstupy ovládací aplikace. V závěru práce je shrnut stav a možnosti aplikace, stejně jako další směřování vývoje.

Klíčová slova

Robotická ruka, Linux, Arduino, Aplikace, Ovládání, Servo

Abstract

This work deals with a problem of designing an application for controlling robotic manipulator using the programming language C++. In the introduction, current hardware and software solutions, available at the market, are described. In the next part a selection of a suitable platform is solved as well as connecting all system elements together, which means the communication between the controlling board and the controlling PC. Afterwards, an algorithm for the robotic arm controlling and its inclusion in the controlling library is described. Then the work includes a design of the controlling application and details of its implementation using the Qt framework. In the last part, controlling the robotic arm from the user's view and available inputs and outputs of the controlling application are described. In the conclusion of the work, the state of the application is summed up, as well as the next development aim.

Keywords

Robotic arm, Linux, Arduino, Application, Controlling, Servo

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně, s použitím odborné literatury a pramenů uvedených v seznamu, který je součástí této bakalářské práce.

Dále prohlašuji, že veškerý software, použitý při řešení této bakalářské práce, je legální.

V Plzni dne 2.6.2015 Michal Horáček

.....

podpis

Poděkování

Rád bych zde poděkoval vedoucímu práce, Ing. Lukáši Koudelovi za věcné připomínky ke zpracování této práce. Dále bych rád poděkoval Ing. Františku Machovi a Doc. Ing. Pavlu Karbanovi, Ph.D. za konzultace a nápady při vývoji software.

Obsah

1 Motivace a cíle práce

Tato práce vznikla na základě požadavku mít prostředek, který by umožňoval snadnější manipulaci s měřícími sondami při měřeních. Ty obvykle vyžadují dlouhodobou přítomnost obsluhy, či opakované umisťování měřící sondy do přesně definovaných poloh. Hlavním cílem práce je tedy vytvoření systému, který by umožnil:

- umisťovat měřící nástroj do jasně definovaných poloh,
- definovat různé trasy nástroje,
- opakovat stejné pohyby po určitou dobu.

Obsah této práce spočívá v návrhu a implementaci:

1. programu pro mikroprocesor, který bude schopen ovládat servo motory robotického manipulátoru pomocí příkazů z PC posílaných sériovou komunikací,
2. modulu pro komunikaci s deskou Arduino pro OS GNU/Linux, tak aby bylo možné z PC odesílat příkazy do mikroprocesoru,
3. abstrakce robotického manipulátoru, která poslouží ke snadnému ovládání v aplikaci,
4. grafické aplikace, která umožní uživateli robotickou ruku snadno ovládat.

1.1 Existující HW řešení

Na trhu lze najít velké množství různých druhů robotických manipulátorů. Když se podíváme pouze na robotické ruky a úchopy, zjistíme, že většina manipulátorů nabízí 4 stupně volnosti a možnost uchopení předmětů, případně levnější typy jen 3 stupně volnosti.

Významným výrobcem v oblasti menších manipulátorů je společnost Lynxmotion [1], která se zabývá výhradně robotickými manipulátory a jejich řízením. Stejná společnost stojí za aplikací RIOS SSC-32 Arm Control Software, o které bude řeč v následující kapitole. Nabízené produkty jsou velice slušně provedené, avšak pro větší aplikace nevhodné, hlavně z důvodu menších rozměrů a menší možné zátěže ramen.

Podobně zaměřená společnost, která se zabývá menšími roboty, je OWI [2]. Ta nabízí několik zajímavých řešení robotických rukou, které jsou však zaměřené spíše na vzdělávání o robotice, či jako pouhá hračka.

Dalším zajímavým výrobcem je společnost Commonplace robotics, GmbH [3]. Ta se zabývá o něco většími manipulátory, zaměřenými na použití ve školách a na univerzitách. Ty jsou vhodné k přemisťování i větších a těžších předmětů, než jsou měřící sondy. Pro naši aplikaci by bylo možné je využít, avšak pro malé měřící sondy jsou tyto manipulátory relativně předimenzované. Jejich cena je také několikanásobně vyšší než v případě společnosti Lynxmotion.

Dále na trhu najdeme společnost Invenscience LC [4], která nabízí sofistikovanější manipulátory, vybavené třeba i senzory polohy. Tato řešení však značně přesahují požadavky, které byly zmíněny v úvodní části práce.

[1] http://www.lynxmotion.com
[2] http://www.owirobot.com
[3] http://commonplace-robotics.com
[4] http://invenscience.com

1.2 Existující software

Jak již bylo zmíněno, společnost Lynxmotion dodává ke svým manipulátorům aplikaci RIOS SSC-32 Arm Control Software [5], která je dostupná pouze pro systém Microsoft Windows. Tato aplikace umožňuje plnou konfiguraci robotické ruky, ovládání ruky v reálném čase i programování jednoduchých sekvencí pohybů. Nabízí také grafický náhled pohybu ruky či vykreslování různých veličin do grafů. Aplikace je bohužel uzavřená a svázaná s produkty firmy Lynxmotion.

Podobně i společnost Commonplace robotics, GmbH nabízí ovládací software ke svým produktům. K dispozici je ovládací konzole TinyCtrl [6], postavená na Linuxu, která umožňuje jednoduché programování pohybu ruky včetně 3D náhledu. Druhou aplikací je CPRog [7], určená pro PC a operační systém Microsoft Windows. CPRog umožňuje díky digitálním vstupům a výstupům komunikaci s průmyslovými PLC. Poslední nabízenou aplikací je RobotExpress [8], také jen pro Microsoft Windows. Aplikace umožňuje komplexní ovládání i většího počtu robotů, včetně detekce kolizí, 3D simulace a generování strojového kódu pro roboty.

Zmíněná komerční řešení jsou však uzavřená a svázaná s produkty dané společnosti. Jelikož byla pro tuto práci vybrána robotická ruka společnost Lynxmotion (viz kapitola 1.3 Výběr platformy), nabízí se možnost využít software RIOS SSC-32 Arm Control Software. Tato aplikace by však postrádala jakoukoli možnost modernizace či implementace nových funkcí. Druhým problémem je, že aplikace komunikuje s ovládacím hardware, který je opět specifický pro konkrétní manipulátor.

1.3 Výběr platformy

1.3.1 Hardware

Pro ovládání robotického manipulátoru byla zvolena deska Arduino Duemilanove [9]. Je vybavena mikroprocesorem ATMega 328p, který disponuje 6 PWM výstupy, potřebnými k ovládání servo motorů robotického manipulátoru. Projekt Arduino také zahrnuje knihovny v jazyce C++, které zjednodušují přístup k řízení. Ty zahrnují všechny běžné úkoly, které lze běžně potřebovat. Od jednoduchých funkcí pro zápis logických hodnot na jednotlivé výstupy, přes generování PWM o zvolené frekvenci až po komunikaci přes UART. Jak po hardwarové stránce, tak po softwarové se jedná o příjemně jednoduchou platformu, vhodnou především pro jednodušší projekty, kterým ovládání jednoho robotického manipulátoru je.

Samotná robotická ruka byla vybrána od společnost Lynxmotion, konkrétně typ AL5A [10], se 4 stupni volnosti a ovládaným úchopem. Střední dosah je přibližně 14.6 cm (5.75“). Obsahuje celkem 5 servo motorů, 3x HS-422, 1x HS-485HB a 1x HS-755HB. Tento typ je dodáván prakticky ve třech různých variantách, a sice jako samotný manipulátor, nebo s přidanou ovládací elektronikou, a nebo ještě s dodaným ovládacím software. Výběrem samotného manipulátoru bez elektroniky nebo ovládacího software chceme docílit hlavně vytvoření neproprietárního řešení, které by umožňovalo větší flexibilitu a případné propojení s jinými systémy.

[5] http://www.lynxmotion.com/p-419-lynxmotion-rios-ssc-32-arm-control-software.aspx
[6] http://commonplace-robotics.com/products/tinyctrl.html
[7] http://commonplace-robotics.com/products/cprog.html
[8] http://commonplace-robotics.com/products/robotexpress.html
[9] http://www.arduino.cc/en/Main/ArduinoBoardDuemilanove
[10] http://www.lynxmotion.com/c-124-al5a.aspx

Obrázek 1.1: Robotická ruka AL5A

Zdroj: http://www.lynxmotion.com/images/product/large/AL5ACN-KT-32U.jpg

1.3.2 Software

S deskou Arduino na druhé straně komunikuje PC, kde postačí jeden USB port. Ovládací aplikace určená pro ovládání robotického manipulátoru je vyvíjena s myšlenkou nasazení na PC s operačním systémem GNU/Linux, od čehož se odvíjí především nejnižší úroveň aplikace, tedy komunikace s hardware. Pro ovládání hardware je napsána poměrně vysokoúrovňová vrstva, která zobecňuje přístup k manipulátoru, tak aby ji šlo snadno integrovat nejen do grafické aplikace, ale i do jiných projektů.

Pro samotné grafické rozhraní aplikace byl zvolen Qt framework [11]. Jedním z důvodů je přenositelnost aplikace mezi operačními systémy GNU/Linux a Microsoft Windows. Qt framework existuje pro oba systémy a případné rozšíření ovládací aplikace o podporu dalších systémů by tak mělo být velice snadné.

[11] http://www.qt.io

2 Nízkoúrovňového ovládání manipulátoru

2.1 Komunikace přes UART

Komunikace mezi deskou Arduino a ovládacím PC probíhá pomocí sériového portu, respektive jeho emulace v systému GNU/Linux. K jeho ovládání je zapotřebí standardních knihoven `termios.h`, `fnctl.h` a `unistd.h`. Při zvolení možnosti připojení ruky dojde k otevření sériového portu a jeho resetování, které se provádí nastavením rychlosti přenosu na 0 Bd. Tím se nastaví ovládací signál DTR (Data Terminal Ready) na 0 a Arduino se resetuje. Poté lze rychlost nastavit libovolně podle požadavků, ačkoli je rychlost přenosu v programu v mikroprocesoru daná pevně. Pro její změnu by bylo nutné program znovu zkompilovat a nahrát do mikroprocesoru.

Po resetování Arduino se spouští funkce `setup()`. Ta je specifická pro projekt Arduino a provádí úkoly, které předcházejí hlavní smyčce programu. V případě robotického manipulátoru se jedná o nastavení do neutrální polohy a inicializaci sériového portu. Na sériový kanál je poté vyslán znak 1, čímž se dává najevo, že je manipulátor připraven k přijímání pokynů.

Hlavní smyčka programu je v projektu Arduino zastoupena funkcí `loop()` namísto obvyklé funkce `main()`. V této funkci se čte sériový kanál a pokud přijde příkaz ve formátu `<číslo servo motoru>,<úhel>` (např. pro servo motor 1 a 90° by vypadal příkaz takto: `1,90`), program plynule nastaví servo motor do požadovaného úhlu. Jakmile pohyb skončí, je směrem k PC vyslán znak `0`, který označuje úspěšné dosažení požadovaného úhlu. Dokud aplikace na PC nepřijme tento znak, nevysílá žádné další příkazy. Jedná se o blokující operaci, při které celá aplikace čeká na dokončení. Aby se aplikace nezastavila úplně v případě selhání robotické ruky, je tato situace navíc ošetřena jistou časovou prodlevou, tzv. timeout. Pokud ani po této časové periodě nepřijde znak 0, čekání se přeruší a ovládací aplikaci se signalizuje, že došlo k chybě.

2.2 Způsob řízení

Základním požadavkem na ovládání manipulátoru je postavení manipulátoru do požadované polohy v kartézském souřadném systému. V ovládací aplikaci je potřeba zadávat dráhu pohybu ve všech třech osách a pro usnadnění obsluhy manipulátoru je vhodné, aby šlo zadat pohyb z bodu do bodu. Z tohoto důvodu je nutné, aby se byl manipulátor schopný přesunout do jiné polohy, nezávisle na současné poloze, co nejkratší cestou, a zároveň, aby byl pohyb plynulý.

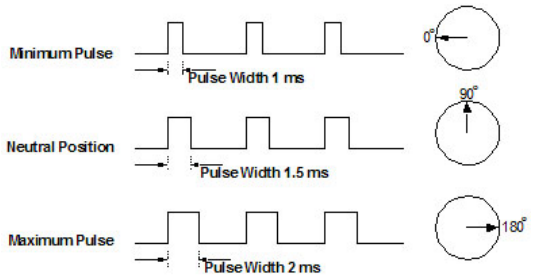
Ovládání servo motorů manipulátoru je však založeno na vztahu šířky pulzu PWM a úhlu natočení osy motoru. Knihovna Arduino umožňuje pomocí metody

```
void write(int angle)
```

třídy `Servo` nastavit úhel natočení servo motoru v rozsahu 0 až 180 stupňů, kde 90 je středová poloha. Pokud je daná hodnota větší než 200, funkce s ní zachází jako s šířkou pulsu v ms. Při přijetí příkazu z PC na nastavení servo motoru se zavolá funkce, která v cyklu postupně nastavuje úhel servo motoru po jednom stupni až dosáhne požadované polohy.

2.2.1 Ovládání servo motoru

Servo motory jsou ovládány šířkou pulsu stejnosměrného signálu. Pro tento signál je z motoru vyveden signálový vodič, který vede do řídícího obvodu servo motoru. Ačkoli se délka pulsu různí napříč motory různých výrobců, všechny mají určenou délku pulsu pro neutrální polohu (obvykle 1.5 ms). Příklad odezvy na řídící signál ukazuje obrázek 2.1.



Obrázek 2.1: Odezvy servo motoru na různé šířky řídícího pulsu

Zdroj: https://www.servocity.com/assets/images/Untitled-165.jpg

Když servo přijme signál k pohybu, nastaví se do odpovídající polohy a setrvá v ní. Motor se bude snažit v poloze setrvat i v případě, že na něj bude působit externí síla. Maximální síla, kterou lze motor zatížit, je k nalezení v katalogovém listu každého servo motoru.

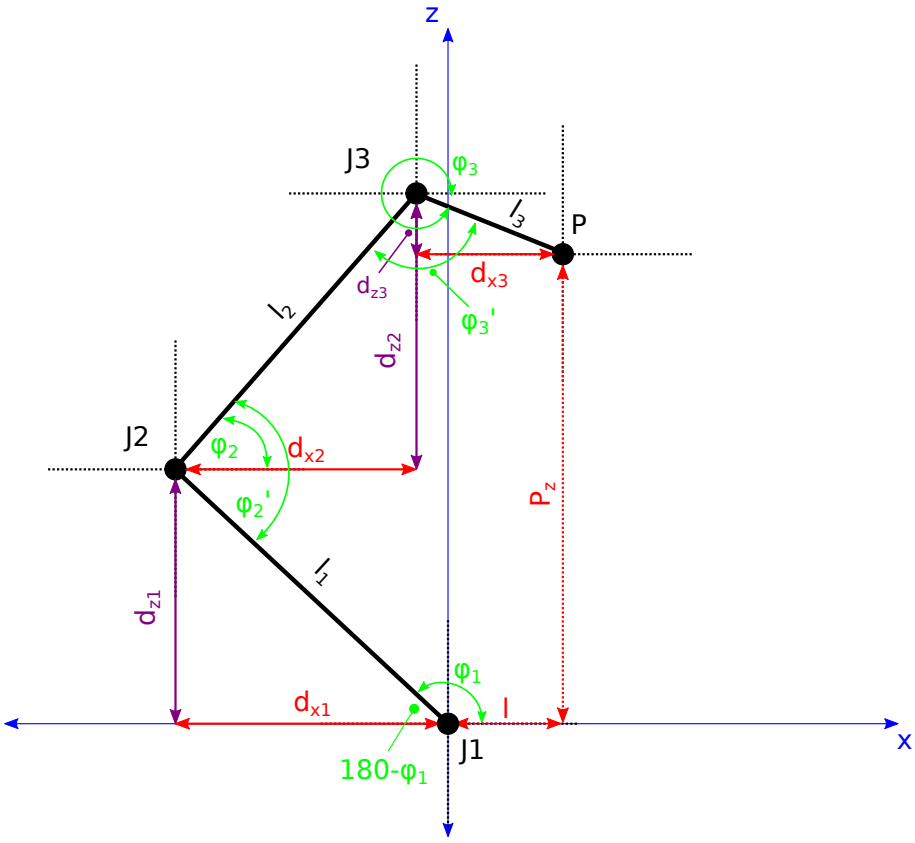
2.3 Polohování robotické ruky

2.3.1 Výpočet polohy ruky z úhlů servo motorů

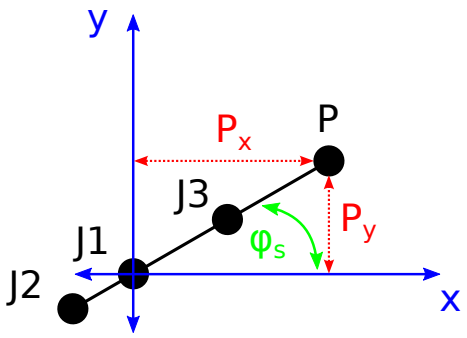
Třída `RoboticArm` obsahuje privátní metodu

```
Point end_point(int S1, int S2, int S3, int S0)
```

která z úhlů servo motorů vypočítá pozici koncového bodu ruky v cm. Tato metoda sama o sobě nemá žádné použití, avšak je základem pro zpětný výpočet úhlů ze souřadnic bodu.



Obrázek 2.2: Ruka - pohled z boku



Obrázek 2.3: Ruka - pohled shora

Ze znalosti úhlů lze snadno určit vzdálenost mezi základnou a koncovým bodem ruky l:

$$l = l_3 \cdot \cos(\varphi_3) + l_2 \cdot \cos(\varphi_2) + l_1 \cdot \cos(\varphi_1)$$

Stejně jako výšku koncového bodu od základny:

$$P_z = l_3 \cdot \sin(\varphi_3) + l_2 \cdot \sin(\varphi_2) + l_1 \cdot \sin(\varphi_1)$$

případně od země:

$$P_z^{'} = l_3 \cdot \sin(\varphi_3) + l_2 \cdot \sin(\varphi_2) + l_1 \cdot \sin(\varphi_1) + z_0$$

kde z_0 je výška základny.

Poté lze ze znalosti natočení základny určit skutečnou polohu bodu:

$$P_x = l \cdot \cos(\varphi_s)$$

$$P_y = l \cdot \sin(\varphi_s)$$

Pro konkrétní ruku je pak ještě třeba před použitím jakéhokoliv úhlu zvážit posunutí, tedy situaci, kdy je servo motor o určitý úhel natočen od základní polohy.

2.3.2 Zpětné určení úhlů servo motorů

Výpočet úhlů je založen na výpočtu z předešlé kapitoly. Vždy se spočítá vzdálenost koncového bodu od základny ruky:

$$l = \sqrt{P_x^2 + P_y^2}$$

Na základě této vzdálenosti lze vypočítat úhel natočení základny:

$$\varphi_s = \arccos\left(\frac{P_x}{l}\right)$$

Opět je zde potřeba pro daný případ přičíst či odečíst posunutí od základní polohy.

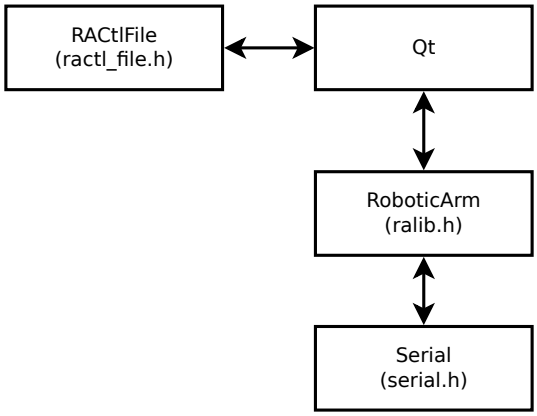
Další úhly se postupně odhadují. V první iteraci se náhodně zvolí úhly v širším rozmezí a vypočítá se poloha odhadnutého koncového bodu P'. Mezi požadovaným bodem P a odhadnutým P' bude ležet určitá vzdálenost, kterou určíme následovně:

$$d = \sqrt{(P_x - P'_x)^2 + (P_y - P'_y)^2 + (P_z - P'_z)^2}$$

Z množiny takto náhodně vybraných bodů se vybere ten, který má vzdálenost k požadovanému bodu d nejmenší. V další iteraci se použijí náhodné úhly z minulé iterace, takže je možné rozmezí pro odhad významně zmenšit. Poté se algoritmus opakuje do té doby, než vzdálenost d dosáhne dostatečně nízké hodnoty, což je v našem případě 10 mm.

2.4 Struktura knihovny

Knihovna je rozdělena do několika souborů, jejichž přehled ukazuje obrázek 2.4. Detailní dokumentace je přílohou této práce a z důvodu univerzality je vedena v anglickém jazyce.



Obrázek 2.4: Struktura knihovny

2.4.1 Komunikace s UART

V první řadě se podívejme na soubor `serial.h`. Ten definuje třídu `Serial`, která obstarává komunikace přes UART. Důležitý je konstruktor třídy:

```
Serial(std::string device, int baudrate)
```

argument `device` je úplná cesta k zařízení ve složce `/dev`. Tento soubor by měl systém automaticky vytvořit po připojení desky Arduino k PC. Druhý argument je samozřejmě rychlost přenosu v Bd. Ta může nabýt velikostí 4800, 9600, 19200 a 38400 Bd, nebo 0 Bd pro resetování kanálu.

Spuštění komunikace se provádí pomocí metody `start()`, která vrací buďto 0 v případě úspěšného navázání komunikace nebo 1 v případě neúspěchu.

Třída umožňuje čtení pomocí metody

```
char readbyte()
```

která přečte pouze jeden znak, nebo

```
std::string readline(int length)
```

která přečte `length` znaků. Zápis je pak možný pomocí metody

```
void writeline(std::string line)
```

2.4.2 Abstrakce robotické ruky

Soubor `ralib.h` definuje namespace RA (zkratka od Robotic Arm), do kterého spadá třída `RoboticArm`, `Point` a třída `RACtlFile`, která je popsána v kapitole 3.3 Vstupy a výstupy.

Nejprve vezměme třídu `Point`. Ta je ve své podstatě pouze datovým typem reprezentujícím bod v kartézských souřadnicích. Má tedy parametry `x`, `y` a `z` typu `float`, které lze nastavit v konstruktoru. Ten je navíc přetížený, aby vstupní parametry mohly být i jiných typů než `float`, a to `int` a `double`. V každém případě jsou tyto parametry přetypovány na `float`.

Soubor `ralib.h` dále definuje třídu `RoboticArm`, která má konstruktor podobný třídě `Serial`:

```
RoboticArm(string device_path, int baudrate)
```

V konstruktoru se parametry pouze předají do objektu třídy a teprve potom je možné ruku zprovoznit zavoláním metody

```
int connect()
```

Zde se inicializuje sériový port, počká se, až z mikroprocesoru desky Arduino přijde znak 1 a poté je možné začít ruku ovládat. Pro odpojení ruky lze použít metodu

```
void disconnect()
```

nebo smazat objekt `RoboticArm`, kde se v destruktoru ruka a sériový kanál odpojí automaticky.

Pro zjištění aktuální polohy je možné zavolat metodu

```
Point position()
```

která vrací polohu koncového bodu ruky.

Nejdůležitější metodou je však

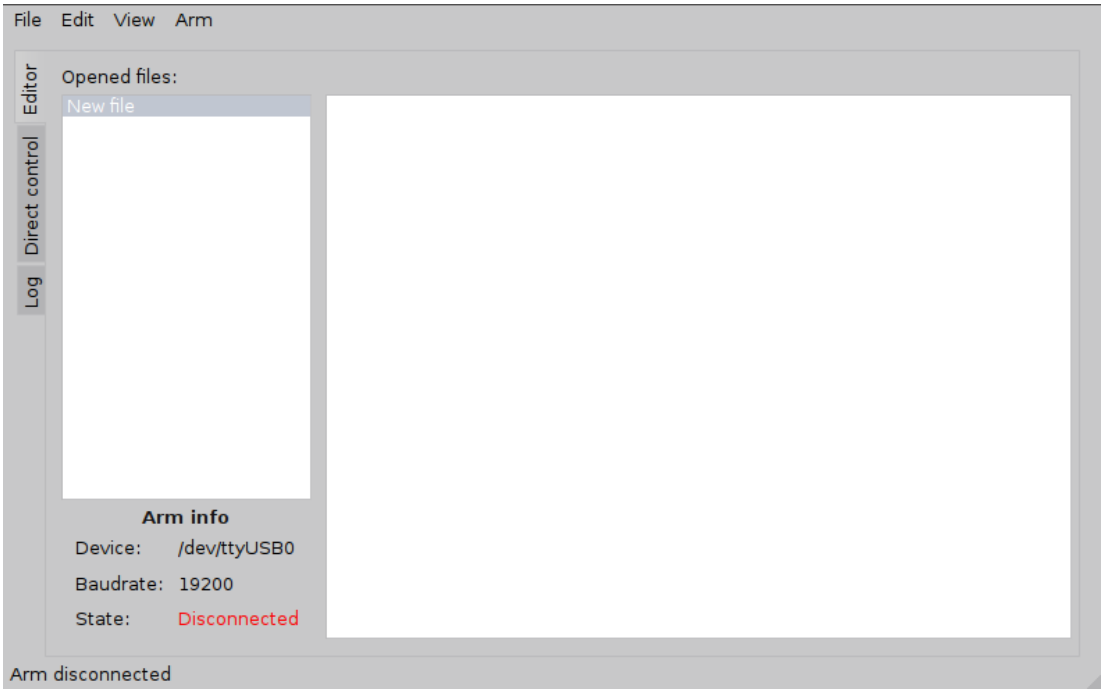
```
int move_to(Point target)
```

Tato metoda zajistí nastavení polohy ruky do požadovaného bodu (`target`). Detailní popis, jak funguje nastavování polohy, je popsán v kapitole 2.3 Polohování robotické ruky.

3 Návrh grafické aplikace

Jak již bylo řečeno v úvodní kapitole, ovládací aplikace by měla uživateli umožňovat snadné ovládání robotické ruky, což zahrnuje:

- konfiguraci komunikace s rukou,
- připojení a odpojení ruky,
- signalizaci stavu ruky a ošetření možných chyb komunikace,
- přímé odeslání příkazu na nastavení do určité polohy,
- definování vlastní sekvence pohybů.



Obrázek 3.1: Úvodní okno ovládací aplikace

3.1 Začlenění knihovny

Knihovna se do grafické aplikace přidá připojením souboru `ralib.h`. Poté se vytvoří objekt `RoboticArm`, pokud je uživatelem zvolena možnost připojení robotické ruky, a je možné začít zadávat pokyny pro pohyb.

Veškeré další ovládání pak probíhá pomocí objektu `RoboticArm` a jeho metody `move_to`. Samotný objekt `RoboticArm` neposkytuje žádné další metody pro pohyb, jako třeba pohyb po čáře. Důvodem je zde právě začlenění do dalších aplikací. Jelikož je k dispozici pouze pohyb do jednoho bodu, je velice jednoduché knihovnu začlenit. Pro složitější pohyby je možné buďto třídu zdědit a stavět na ní nebo komplexnější pohyby vypočítat předem a poté volat pouze metodu pro přesunutí ruky do jednoho bodu.

Pro zjištění aktuální polohy ruky, jak již bylo řečeno, lze využít metody `position()`. Tím lze v aplikaci, postavené nad knihovnou ralib, pracovat s polohou ruky a předávat tak informace uživateli.

3.2 Ovládání aplikace

3.2.1 Jazyk aplikace

Při spuštění aplikace se načítá informace o lokalizaci. Pomocí třídy `QLocale` lze z operačního systému zjistit jazyk, který uživatel používá a poté pomocí třídy `QTranslator` k aplikaci připojit daný soubor s překlady. Qt využívá vlastní formát lokalizačních souborů, které mají příponu `qm` a jedná se o binární soubory, které lze vygenerovat programem `lrelease` ze souborů s příponou `ts`. To jsou XML popisující jednotlivé texty aplikace viditelné pro uživatele. Qt navíc nabízí aplikaci Qt Linguist, určenou ke snadné editaci `ts` souborů a k jejich následnému převedení do formátu `qm`. Jakmile jsou tyto soubory vygenerované a dostupné aplikaci, všechny texty označené k překladu jsou automaticky přeloženy.

Aby byl text přeložitelný, je třeba použít funkci `tr`, která je definována jako:

```
static inline QString tr(const char *s, const char *c = 0)

static inline QString tr(const char *s, const char *c, int n)
```

kde `s` je text určený k překladu. Argument `c` je identifikátor objektu pro překladatele, jelikož v různých jazycích může mít stejné slovo jiný tvar v závislosti na kontextu. Poslední argument `n` určuje počet objektů, ke kterým se text vztahuje, aby mohl překladatel slovo, či slovní spojení, správně skloňovat.

3.2.2 Nastavení aplikace

Pro ukládání a načítání nastavení se v Qt využívá třída `QSettings`. Aby bylo nastavení viditelné pro všechny části aplikace, je třeba v souboru `main.cpp` specifikovat název organizace a název aplikace:

```
QCoreApplication::setOrganizationName("organizace");

QCoreApplication::setApplicationName("aplikace");
```

Poté se ve složce aktuálního uživatele vytvoří odpovídající cesta a soubor s nastavením (`*.conf`). V OS GNU/Linux to povede na `~/.config/organizace/aplikace.conf`. Soubor s nastavením je strukturou shodný s INI soubory, které se pro ukládání nastavení aplikací běžně používají.

Aplikace umožňuje nastavit parametry sériového portu, tedy jeho umístění v adresáři `/dev` a jeho rychlost. Další důležitou možností je nastavení rozměrů ruky, které jsou potřeba pro výpočet polohy.

3.2.3 Konfigurace sériové komunikace

V nastavení aplikace lze v záložce Sériový port nastavit cestu k souboru, který odpovídá sériovému portu (často `/dev/ttyACM*` nebo `/dev/ttyUSB*`). Kromě toho lze nastavit rychlost komunikace v Bd. Jak bylo ovšem zmíněno v kapitole 2.1 Komunikace přes UART, změnu nelze provést bez zásahu do programu v mikroprocesoru, který ruku ovládá. Možnost nastavit rychlost komunikace je zde právě kvůli možné změně na straně robotické ruky. Kdyby k takové změně došlo, je třeba na ni pružně reagovat, tedy bez nutnosti upravovat zdrojový kód ovládací aplikace.

3.2.4 Ovládání ruky

K ovládání ruky je potřeba nejprve zvolit možnost jejího připojení. Tím se vytvoří instance `RoboticArm`, předají se jí parametry nastavené uživatelem a některé prvky aplikace se změní, aby připojení ruky patřičně reflektovaly.

Jakmile je ruka připojená, lze ji ovládat v zásadě dvěma způsoby. Buďto pomocí přímého ovládání nebo pomocí editoru tras. Oba dva způsoby jsou podrobněji popsány v následujících kapitolách.

Po ukončení práce s rukou je možné ji opět odpojit, což vede ke změně ovládacích prvků aplikace a smazání objektu `RoboticArm`. Odpojení ruky se provádí automaticky při ukončení aplikace.

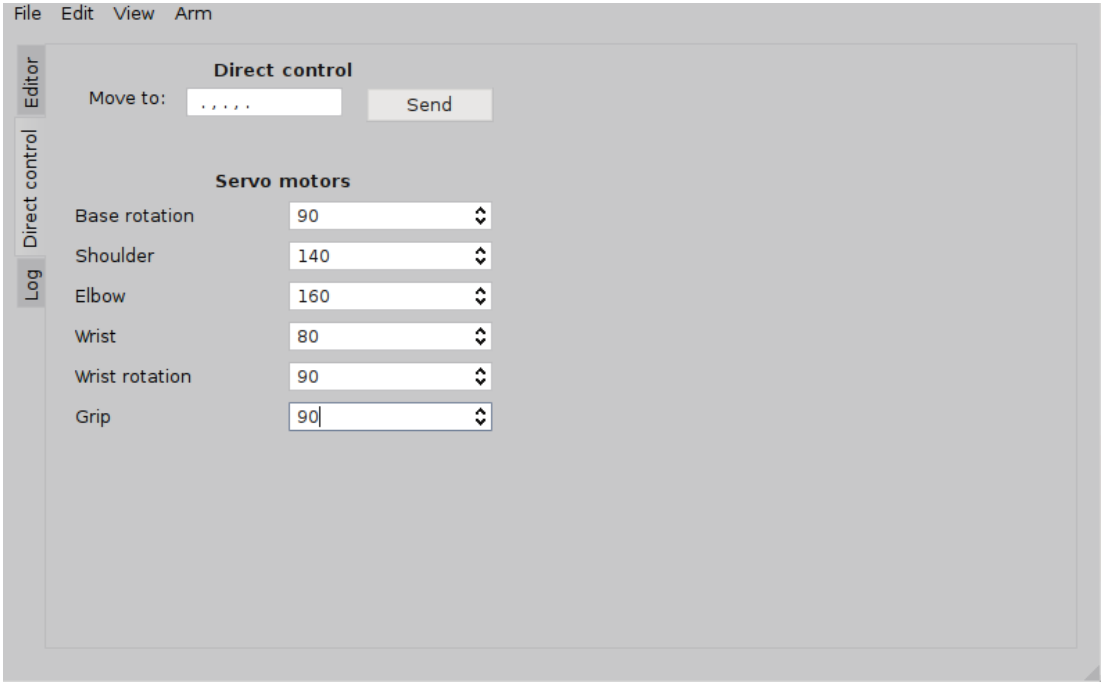
3.2.5 Přímé ovládání ruky

Přímé ovládání umožňuje uživateli polohovat ruku buďto do konkrétního bodu bez použití editoru tras, nebo ovládat jednotlivé servo motory, každý zvlášť. K tomu slouží šest vstupních polí typu `spinBox`. Ty emitují signál `valueChanged(int)`, který je zaveden do objektu `QSignalMapper`. Samotný `QSignalMapper` je pak zaveden k funkci

```
void MainWindow::servoAngleSet(int servo_n)
```

která nastavuje servo motor s číslem `servo_n` na hodnotu, která je momentálně nastavená ve vstupním poli. Tím se zajistí, že lze využít pouze jednu funkci pro všechny servo motory namísto jedné funkce pro každé servo.

Ukázku přímého ovládání ruky ukazuje obrázek 3.2.



Obrázek 3.2: Přímé ovládání ruky

3.2.6 Editor tras

Editor tras je oproti přímému ovládání pohodlnější způsob práce s rukou, neboť umožňuje definování libovolně dlouhé sekvence pohybů. Pro psaní příkazů je použita komponenta `QTextWidget`. Ta, kromě samotného textu, podporuje i formátování, které je využito k obarvení syntaxe zadaných příkazů.

Aby bylo obarvení možné, je potřeba vytvořit novou třídu, která dědí z třídy `QSyntaxHighlighter`. V tomto projektu se jedná o soubor `ractl_highlighter.h` a třídu `RACtlHighlighter` v něm definovanou. Ta má následující konstruktor:

```
RACtlHighlighter(QTextDocument* doc)
```

Zde se objektu předá ukazatel na dokument, na který se má obarvení aplikovat. Použití pak vypadá takto:

```
this->highlighter = new RACtlHighlighter(this->ui->textEdit->document())
```

tím se dokument editoru předá zvýrazňovači a obarvování pak probíhá automaticky, kdy je potřeba.

Podívejme se nyní na to, jak třída `RACtlHighlighter` vypadá uvnitř. Vše se odehrává v podstatě pouze v jedné metodě:

```
void highlightBlock(const QString &text)
```

kde vystupuje argument `text` typu `QString`. To je libovolný text, na který se mají zvýrazňovací pravidla vztahovat. Samotný výběr částí textu se provádí pomocí regulárních výrazů. Pokud takovému výrazu některá část textu vyhovuje, je na ní aplikován styl textu definovaný objektem třídy `QTextCharFormat`.

Aplikace zvládá obarvování klíčových slov, respektive příkazů pro pohyb, čísel a komentářů, které může uživatel do souboru libovolně vkládat.

3.3 Vstupy a výstupy

Pro práci se soubory je součástí aplikace soubor `ractl_file.h`. Zde se k namespace `RA` přidává třída `RACtlFile`, která zastupuje právě jeden textový soubor. Otevření souboru se provádí metodou:

```
int open(std::string filename)
```

která přijímá parametr `filename` jako název otevíraného souboru, která může být jak absolutní, tak relativní k umístění aplikace. Pokud soubor neexistuje, není vytvořen, avšak objekt v paměti setrvá kvůli možnému zápisu. Ten se provádí pomocí metody:

```
int save(std::string filename=std::string(""))
```

Zde vidíme opět parametr názvu souboru, avšak jako volitelnou hodnotu. Pokud název souboru není uveden, je použit název uvedený při otevření souboru metodou `open(filename)`. Pokud by byla metoda `write` zavolána a název souboru by nebyl uveden při otevření souboru, nebo při zápisu do něj, pak `write` vrací hodnotu 1 jako chybu.

Třída používá soubory kódované v UTF-8. Aby ostatní aplikace (textové editory, internetové prohlížeče) poznaly, že se jedná o UTF-8 (resp. i ostatní znakové sady unicode), zapisuje se na začátek souboru tzv. BOM (byte order mark). Jeho forma je 0xEF, 0xBB, 0xBF.

3.3.1 Trasy robotické ruky

Aplikace používá vlastní formát souborů pro popis pohybu ruky. Jedná se o textové soubory s příponou `*.path`. Jde o sérii příkazů, které umožňují programování pohybu ruky, stejně jako běžná komerční řešení dostupná na trhu.

Záhlaví souboru

Zde existují klíčová slova, jejichž účel je pouze informativní a do ovládání ruky nijak nezasahují. Tato klíčová slova mají v editoru specifické obarvení, odlišné od ostatních příkazů. Do souboru lze specifikovat:

`@author` - pro informaci o autoru daného souboru

`@description` - pro popis, k čemu je soubor určen

`@version` - pro anotaci verze daného souboru

Komentáře

Pro přehlednost souborů, zvláště pokud se skládají z většího množství příkazů, umožňuje editor psát komentáře, které jsou obarveny méně výraznou šedou barvou. Jejich formát může být dvojí:

```
#komentář
//komentář
```

Příkazy pohybu

Příkazů pro pohyb je celá řada. Tím základním je umístění ruky do určitého bodu:

```
target [x, y, z]
```

```
point [x, y, z]
```

Oba příkazy jsou totožné a v závorce je předáván koncový bod ruky. Všechny souřadnice jsou v cm.

Dalším důležitým příkazem je pohyb po přímce:

```
line [x1, y1, z1] [x2, y2, z2] n delay
```

kde se uvádí počáteční a koncový bod, počet bodů na dané přímce (`n`) a doba, po kterou se pohyb v každém bodě přímky zastaví (`delay`). Parametr `delay` je uváděn v ms a je volitelný. Pokud není uveden, předpokládá se 0 ms.

Pro samotné vyčkání mezi pohyby lze využít příkaz pozastavení. Opět zde existuje více názvů pro stejnou funkci:

```
delay t
```

kde `t` je opět čas v ms.

Druhá možnost je využít příkaz

```
sleep t
```

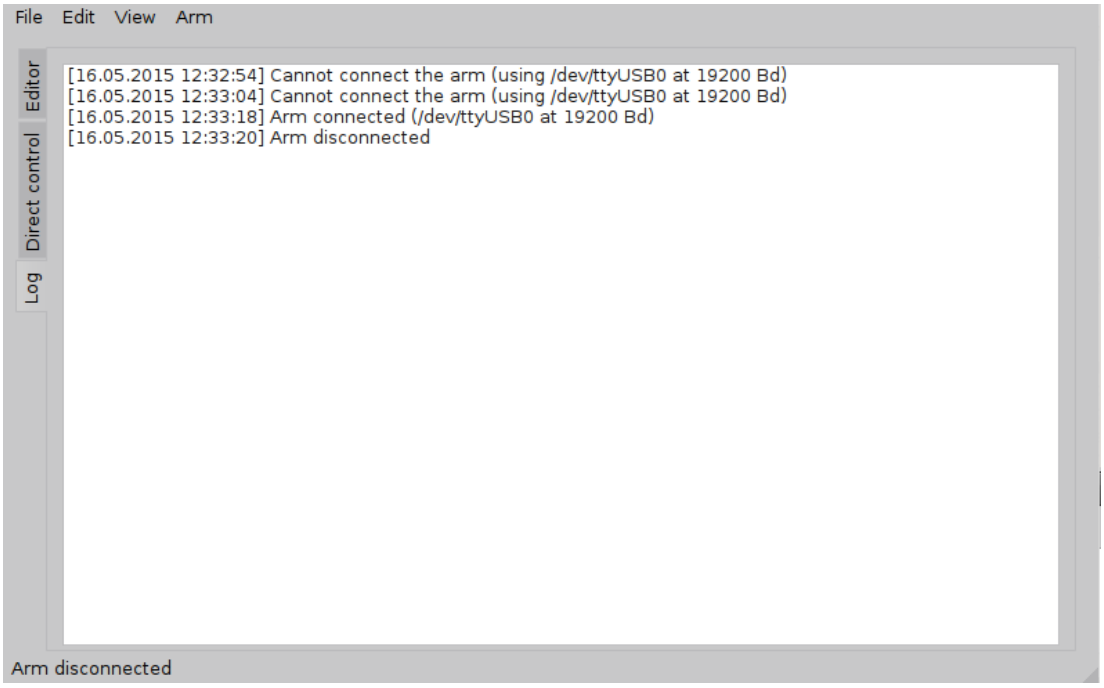
který oproti příkazu `delay` přijímá parametr v sekundách.

Pro uvedení ruky do počáteční polohy slouží příkaz

```
park
```

3.3.2 Záznamy z běhu aplikace

Aplikace ukládá záznamy o své činnosti do textových souborů ve složce `logs`. Automaticky se postará o vytvoření dané složky i souboru. Stejný obsah, jako se ukládá do souboru, se také zobrazuje v aplikaci v záložce Log. Ukázku takového záznamu ukazuje obrázek 3.3.



Obrázek 3.3: Ukázka záznamu činností prováděných aplikací

4 Závěr

Cílem práce bylo navrhnout a vytvořit aplikaci pro ovládání robotické ruky. V době psaní této práce byla aplikace schopna základní konfigurace robotické ruky prostřednictvím příslušného dialogu. Aplikace umožňovala ovládání ruky jak manuálně díky záložce Přímé ovládání, tak i automatizováním pohybů vestavěným editorem a příkazy pro pohyb. Samozřejmostí jsou základní funkce pro práci se soubory, tedy jejich ukládání a načítání, možnost přepínání mezi vícero otevřenými soubory a zvýraznění klíčových slov v nich obsažených. V neposlední řadě bylo k dispozici ukládání záznamu z běhu aplikace.

Primární platforma, pro kterou aplikace vznikla, je GNU/Linux. Pro tento systém je aplikace snadno dostupná a její instalace je otázkou několika málo minut. Pro kompilaci ze zdrojových souborů je zapotřebí standardních Linuxových nástrojů, tedy g++, make a samozřejmě knihoven Qt4. Instrukce pro sestavení aplikace jsou dostupné v adresáři se zdrojovými soubory.

Další vývoj aplikace by se měl zaměřit na rozšíření možností pohybů. Pomocí bodů a přímek lze sestavit velké množství různých pohybů, avšak kdyby bylo k dispozici více základních příkazů pohybů, byla by robotická ruka snazší na ovládání.

Změn by také mohl doznat algoritmus pro umisťování robotické ruky. Ačkoli je současné řešení funkční, z hlediska rychlosti není zcela optimální. K tomu se také váže schopnost otáčet zápěstím ruky, kterou by bylo dobré lépe propojit s příkazy pohybu, aby bylo umožněno pohybovat se např. po plášti koule.

Zajímavým vylepšením by byla simulace pohybu ruky, podobně jako to umožňují komerční řešení zmíněná v úvodu práce. Vývoj by se mohl ubírat směrem k OpenGL knihovně a její integrace do Qt. Takové řešení by pak mohlo fungovat jak na operačním systému GNU/Linux, tak i Microsoft Windows, případně dalších. Aplikace by pak mohla umožnit vizuální kontrolu naprogramovaného pohybu ruky ještě před jejím spuštěním.

Dalším vylepšením by mohla být integrace skriptovacího jazyka Python. Umožnilo by to komfortnější ovládání celé aplikace a pravděpodobně také rozšíření o další funkcionalitu.

Pokud bychom zůstali u operačního systému GNU/Linux, pak by stálo za zmínku využití protokolu pro komunikaci mezi aplikacemi, D-Bus. To by mohlo umožnit vzdálený přístup k ovládací aplikaci. Toto řešení by se dalo následně rozšířit na webovou aplikaci, která by umožňovala spouštění měření např. z lokální sítě.

Seznam literatury

[1] Arduino. *Arduino Duemilanove* [online]. [cit. 18.4.2015] Dostupné z: `http://www.arduino.cc/en/Main/ArduinoBoardDuemilanove`

[2] Arduino. *Servo library* [online]. [cit. 18.4.2015] Dostupné z: `http://www.arduino.cc/en/Reference/Servo`

[3] Robotzone, LLC. *How do servos work?* [online]. [cit. 18.4.2015] Dostupné z: `https://www.servocity.com/html/how_do_servos_work_.html`

[4] The Qt Company. *Qt Linguist Manual: Programmers* [online]. [cit. 18.4.2015] Dostupné z: `http://doc.qt.io/qt-4.8/linguist-programmers.html`

[5] The Qt Company. *QSettings Class* [online]. [cit. 18.4.2015] Dostupné z: `http://doc.qt.io/qt-4.8/qsettings.html`

[6] The Qt Company. *QSignalMapper Class* [online]. [cit. 14.5.2015] Dostupné z: `http://doc.qt.io/qt-4.8/qsignalmapper.html`

[7] The Qt Company. *QSyntaxHighlighter Class* [online]. [cit. 19.4.2015] Dostupné z: `http://doc.qt.io/qt-4.8/qsyntaxhighlighter.html`

[8] Unicode, Inc.. *UTF-8, UTF-16, UTF-32 & BOM* [online]. [cit. 18.4.2015] Dostupné z: `http://www.unicode.org/faq/utf_bom.html`

[9] Mark Zehner, Peter Baumann. *Serial Port Programming in Linux* [online]. [cit. 18.4.2015] Dostupné z: `http://trainingkits.gweb.io/serial-linux.html`

[10] Free Software Foundation, Inc.. *The GNU C Library: Creating Directories* [online]. [cit. 19.5.2015] Dostupné z: `https://www.gnu.org/software/libc/manual/html_node/Creating-Directories.html`

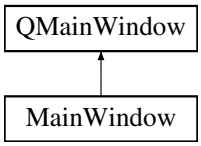
Seznam obrázků

Příloha A - Dokumentace zdrojového kódu

Contents

MainWindow Class Reference

Inheritance diagram for MainWindow:



Public Slots

- void connectArm ()

  Connects the arm and changes UI according to that.
- void disconnectArm ()

  Disconnects the arm and changes UI according to that.
- void sendCommand ()

  Reads the content of lineEdit and uses arm->move_to to send a command.
- void runArm ()

  Processes the command list and runs the arm according to them.
- void showSettings ()

  Shows a settings dialog.
- void openFile ()

  Opens an arm path description file.
- void saveFile ()

  Saves the current file.
- void saveFileAs ()

  Shows te save dialog and saves the path file.
- void closeFile ()

  Closes a path file and changes UI to that.
- void reloadUI ()

  Changes those widgets that are changing after some events.
- void fileChanged ()

  Slot called when the textEdit's text is modified. It updates the file's entries and UI to acknowledge these changes..
- void reloadEditor ()

  Changes widgets belonging to the editor part.
- void fileSelectionChanged ()

  Slot called when the selection in the file list is changed.
- void newFile ()

  Creates a new blank file.
- void switchToEditor ()

  Switches the tabWidget to the editor.
- void switchToDirectControl ()

  Switches the tabWidget to the direct control tab.
- void **logMessage** (QString message)
- void **servoAngleSet** (int servo_n)

Public Member Functions

- **MainWindow** (QWidget ∗parent=0)
- void **saveSettings** ()
- void **loadSettings** ()
- void resizeEvent (QResizeEvent ∗)

  Method called when the main window is being resized. Here it resizes also other widgets.
- void syntaxError (unsigned line_n, string line)

  Displays a syntax error (message box) with a line number, where the error was and the line itself.
- void closeEvent (QCloseEvent ∗)

  Rewritten closeEvent checks all open files if they were changed.

Member Function Documentation

void MainWindow::syntaxError (unsigned *line_n,* string *line*)

Displays a syntax error (message box) with a line number, where the error was and the line itself.

Parameters

line_n	Number of the line in the textEdit where the error occurred.
line	The line that contains an error.

The documentation for this class was generated from the following files:

- mainwindow.h
- mainwindow.cpp

RA::Point Class Reference

Public Member Functions

- **Point** (int x, int y, int z)
- **Point** (float x, float y, float z)
- **Point** (double x, double y, double z)
- std::string to_str ()

  Returns std::string representation of the current point position.

Public Attributes

- float **x**
- float **y**
- float **z**

Member Function Documentation

std::string RA::Point::to_str ( )

Returns std::string representation of the current point position.

Returns

std::string representation of the current point position.

The documentation for this class was generated from the following files:

- ralib.h
- ralib.cpp

RA::RACtlFile Class Reference

Public Member Functions

- int open (string filename)
  Opens the file for reading.
- void close ()
  Closes the file.
- int save (string filename=string(""))
  Saves the current file's content.
- string name ()
  Method to get the file name.

Public Attributes

- string **content**
- bool **is_open**

Member Function Documentation

string RA::RACtlFile::name ( )

Method to get the file name.

Returns

std::string filename of the file's object

int RA::RACtlFile::open (string *filename*)

Opens the file for reading.

Parameters

filename	File name to open.

Returns

0 if the file was opened, 1 if it couldn't be opened.

int RA::RACtlFile::save (string *filename* = `string("")`)

Saves the current file's content.

Parameters

filename	Optional argument. If it's blank, the filename set by open() will be used.

Returns

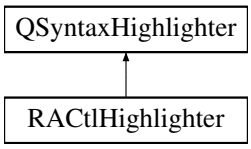
0 if the saving was successful, otherwise returns 1

The documentation for this class was generated from the following files:

- ractl_file.h
- ractl_file.cpp

RACtlHighlighter Class Reference

Inheritance diagram for RACtlHighlighter:



Public Member Functions

- **RACtlHighlighter** (QTextDocument ∗doc)
- void **highlightBlock** (const QString &text)
- void **applyRule** (const QString &text, QString &pattern, QTextCharFormat &format)

The documentation for this class was generated from the following files:

- ractl_highlighter.h
- ractl_highlighter.cpp

RA::RoboticArm Class Reference

Public Member Functions

- **RoboticArm** (string device_path, int baudrate)
- int connect ()

  Connects to the serial port.
- void disconnect ()

  Ends the serial port connection.
- int move_to (Point target)

  RoboticArm::MoveTo sends commands to Arduino board to move the hand's end point to specified coordinates.
- Point position ()

  RoboticArm::Position.
- int set_servo (unsigned servo_number, int angle)

  Sets a servo motor to an angle.

Public Attributes

- float **L1**
- float **L2**
- float **L3**

Member Function Documentation

int RA::RoboticArm::connect ( )

Connects to the serial port.

Returns

0 if connected, 1 if there was an error

int RA::RoboticArm::move_to (Point *target*)

RoboticArm::MoveTo sends commands to Arduino board to move the hand's end point to specified coordinates.

Parameters

target	Target point where the arm should end

Returns

0 if everything went OK, 1 if the arm is not connected, 2 if timeout has been reached

RA::Point RA::RoboticArm::position ( )

RoboticArm::Position.

Returns

Point object - current position of the end point

int RA::RoboticArm::set_servo (unsigned *servo_number,* int *angle*)

Sets a servo motor to an angle.

Parameters

servo_number	- number of servo motor, see the Arduino servo-pin mapping
angle	- in degrees

The documentation for this class was generated from the following files:

- ralib.h
- ralib.cpp

Serial Class Reference

Public Member Functions

- Serial (string device, int baudrate)

- void writeline (string line)

  Sends a line to the serial port. Appends \n.
- void write_text (string text)

  Sends a std::string to the serial port. Then flushes the serial.
- string readline (int length)

  Reads length bytes from the serial port.
- char readbyte ()

  Reads a single byte from the serial port.
- int start ()

  Opens the device.
- void reset ()

  Resets the serial port (close and open again).
- void flush ()

  Flushes the serial port (in and out both).
- void end ()

  Closes the connection by setting baudrate to 0 Bd.

Public Attributes

- bool **is_open** = false

Constructor & Destructor Documentation

Serial::Serial (string *device,* int *baudrate*)

Parameters

device	Full path to the /dev device file.
baudrate	Baudrate of the serial port.

Member Function Documentation

char Serial::readbyte ( )

Reads a single byte from the serial port.

Returns

One char.

string Serial::readline (int *length*)

Reads length bytes from the serial port.

Parameters

length	Number of bytes read.

Returns

std::string of the line read, or blank string if 0 bytes were read.

int Serial::start ( )

Opens the device.

Returns

0 if the device has been opened or 1 if the device coudn't be opened.

void Serial::write_text (string *line*)

Sends a std::string to the serial port. Then flushes the serial.

Parameters

line	String to be send.

void Serial::writeline (string *line*)

Sends a line to the serial port. Appends \n.

Parameters

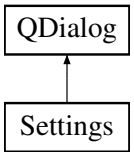
line	String to be send.

The documentation for this class was generated from the following files:

- serial.h
- serial.cpp

Settings Class Reference

Inheritance diagram for Settings:



Public Member Functions

- **Settings** (QWidget ∗parent=0)

The documentation for this class was generated from the following files:

- settings.h
- settings.cpp