VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

PLÁNOVÁNÍ DRÁHY ROBOTICKÉHO RAMENE

ROBOTIC ARM PATH PLANNING

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **TOMÁŠ MAJOR**
AUTHOR

VEDOUCÍ PRÁCE **Ing. JAROSLAV ROZMAN, Ph.D.**
SUPERVISOR

BRNO 2020

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav inteligentních systémů (UITS) Akademický rok 2019/2020

Zadání bakalářské práce

Student: **Major Tomáš**
Program: Informační technologie
Název: **Plánování dráhy robotického ramene**
Robotic Arm Path Planning
Kategorie: Umělá inteligence

Zadání:

1. Seznamte se s knihovnou OpenRAVE/MoveIt, s metodami pro plánování dráhy pro robotická ramena a s ramenem Mitsubishi Melfa.
2. Navrhněte knihovnu, která pomocí MoveIt umožní naplánovat dráhu v simulovaném prostředí pro zvolený model ramene. Dále navrhněte program, který umožní řídit pohyby ramene Mitsubishi Melfa při vykonávání naplánované dráhy v reálném čase.
3. Navrženou knihovnu a program implementujte a zároveň vytvořte demonstrační program, ve kterém ukážete funkčnost knihovny a programu pro řízení v reálném čase.
4. Program otestujte na rameni Mitsubishi Melfa a navrhněte případná vylepšení.

Literatura:

- Howie Choset et al., Principles of Robot Motion, 2005, ISN 0-262-03327-5

Pro udělení zápočtu za první semestr je požadováno:

- První dva body zadání.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Rozman Jaroslav, Ing., Ph.D.**
Vedoucí ústavu: Hanáček Petr, doc. Dr. Ing.
Datum zadání: 1. listopadu 2019
Datum odevzdání: 28. května 2020
Datum schválení: 31. října 2019

Abstrakt

Tato bakalářská práce se zabývá ovládáním robota Mitsubishi Melfa RV-6SL na základě naplánované dráhy knihovnou MoveIt. V této práci se může čtenář dočíst jakým způsobem postupovat a jaké informace potřebuje při snaze zprovoznit robotické rameno v platformě ROS s využitím plánovací knihovny MoveIt. Výsledkem této práce je schopnost ovládat robota Melfa RV-6SL systémem ROS, který je společně s MoveIt v práci krátce představen.

Abstract

This bachelor thesis deals with the control of the Mitsubishi Melfa RV-6SL robot based on the planned path using the MoveIt library. In this thesis the reader can learn how to proceed and what information he needs when he wants to set the robotic arm in motion while using the MoveIt planning library with the ROS platform. The result of this work is the ability to control the Melfa RV-6SL robot with the ROS system, which is together with MoveIt briefly introduced at the beginning of this thesis.

Klíčová slova

MoveIt, robotické rameno, Melfa, ROS-Industrial, ROS, Gazebo, Rviz, plánování dráhy, URDF

Keywords

MoveIt, robotic arm, Melfa, ROS-Industrial, ROS, Gazebo, RViz, path planning, URDF

Citace

MAJOR, Tomáš. *Plánování dráhy robotického ramene*. Brno, 2020. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Jaroslav Rozman, Ph.D.

Plánování dráhy robotického ramene

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Jaroslava Rozmana, Ph.D. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Tomáš Major
28. května 2020

Poděkování

Na tomto místě bych rád poděkoval vedoucímu bakalářské práce Ing. Jaroslavu Rozmanovi, Ph.D., za jeho ochotu, trpělivost a odbornou pomoc, kterou mi při psaní této bakalářské práce poskytl.

Obsah

Kapitola 1

Úvod

Robotizace je v současné době jednou z nejrychleji se vyvíjejících technologických oblastí. Od prvního použití pokrokového výrazu robot, jež bylo prvně vysloveno přesně před sto lety, dosáhla tato oblast velkého rozmachu. Pro roboty je v dnešním moderním světě nespočet možných využití. Jsme v době, kdy lze velké množství typů manuální práce nahradit právě roboty. Postupem času se robotizace stala důležitou součástí běžného pracovního života. Není tedy překvapením, že při návštěvě výrobní továrny či výzkumné laboratoře člověk vždy narazí na tento technologický pokrok.

Cílem práce je ovládat robota Mitsubishi Melfa RV-6SL na základě naplánované dráhy knihovnou MoveIt či OpenRAVE. Z důvodu dostupnosti jsem si pro účely této práce vybral knihovnu MoveIt, která je zároveň uživatelsky přívětivější. MoveIt jako takový zvládne trajektorii ramene naplánovat, avšak nezvládne naplánovaný úkon provést. Proto je robotický operační systém ROS nezbytnou součástí této práce, aby zajistil provedení naplánované dráhy na simulovaném či reálném robotu. V této práci se může čtenář dočíst, jakým způsobem postupovat při snaze zprovoznit robotické rameno v platformě ROS s využitím plánovací knihovny MoveIt.

První část práce je věnována systému ROS a dalším pojmům, které se k němu vztahují, či jsou jinak spjaty s touto prací. Další část je zaměřena na robotické rameno Melfa, kde jsou popsány veškeré potřebné parametry a vytvoření modelu robota ve speciálním formátu nezbytného pro dosažení stanoveného cíle. Dále je vysvětlen plánovací program MoveIt a je popsáno aplikování jeho funkcí pro potřeby této práce. V páté kapitole je popsán volně dostupný ovladač melfa_driver zajišťující komunikaci s kontrolérem robota, který jsem pro účely této práce převzal. V závěrečné fázi práce je zaznamenáno testování dosažených výsledků.

Kapitola 2

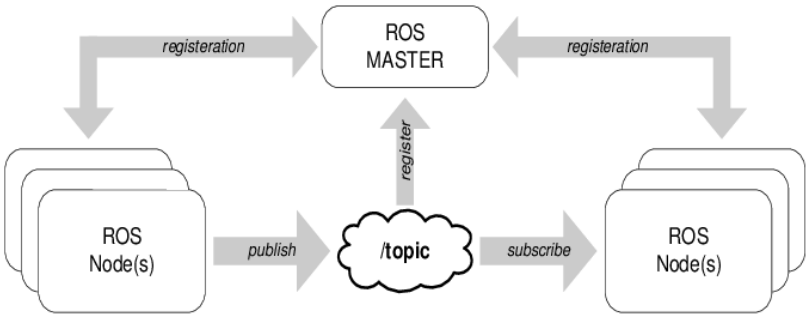
ROS

V této kapitole jsou popsány a vysvětleny knihovny a technologie, jež jsou součástí systému ROS 2.1 a jsou spjaty s touto prací.

2.1 ROS – Robot Operating System

ROS je meta-operační systém poskytující sadu knihoven a nástrojů pro sestavování programů pro roboty. ROS není vlastní operační systém, jsou to knihovny, které dokáží nad operačním systémem efektivně komunikovat s hardwarem. Důvod, proč je tento systém tak rozšířený a stále se rozšiřující je ten, vedle toho, že je open-source, že svojí architekturou 2.1 zvládne ovládat všechny možné roboty od robotických ramen s kamerami, přes drony se všemi různými druhy senzorů, až po lidské roboty s umělou inteligencí. Pro tuto práci je používána poslední stabilní verze ROS Melodic instalovaná na operačním systému Ubuntu 18.04.03.

Hlavními prvky systému ROS jsou uzly (nodes), topiky (topics), zprávy (messages) a služby (services). Služby slouží ke komunikaci, prostřednictvím protokolu RPC, například s uzly na jiném počítači. Uzly se dělí na subscribery a publishery daného topiku. Uzel (samostatný proces) při nějaké změně stavu odešle topiku zprávu nebo ji od něho přijme, a to v závislosti na tom, zda je subscriber nebo publisher. Aby jednotlivé uzly věděly, kde se jaký topik nachází a aby se s ním mohly spojit, je tu hlavní uzel ROS Master, u kterého se jednotlivé uzly a topiky registrují a dále už komunikují pouze mezi sebou. S cílem oddělit jednotlivé operační skupiny pro různé roboty, je ROS strukturován do balíčků (packages). Jeden z takových balíčků je výsledkem programu MoveIt Setup Assistant 4.2 či URDF exporteru 3.4. [18]

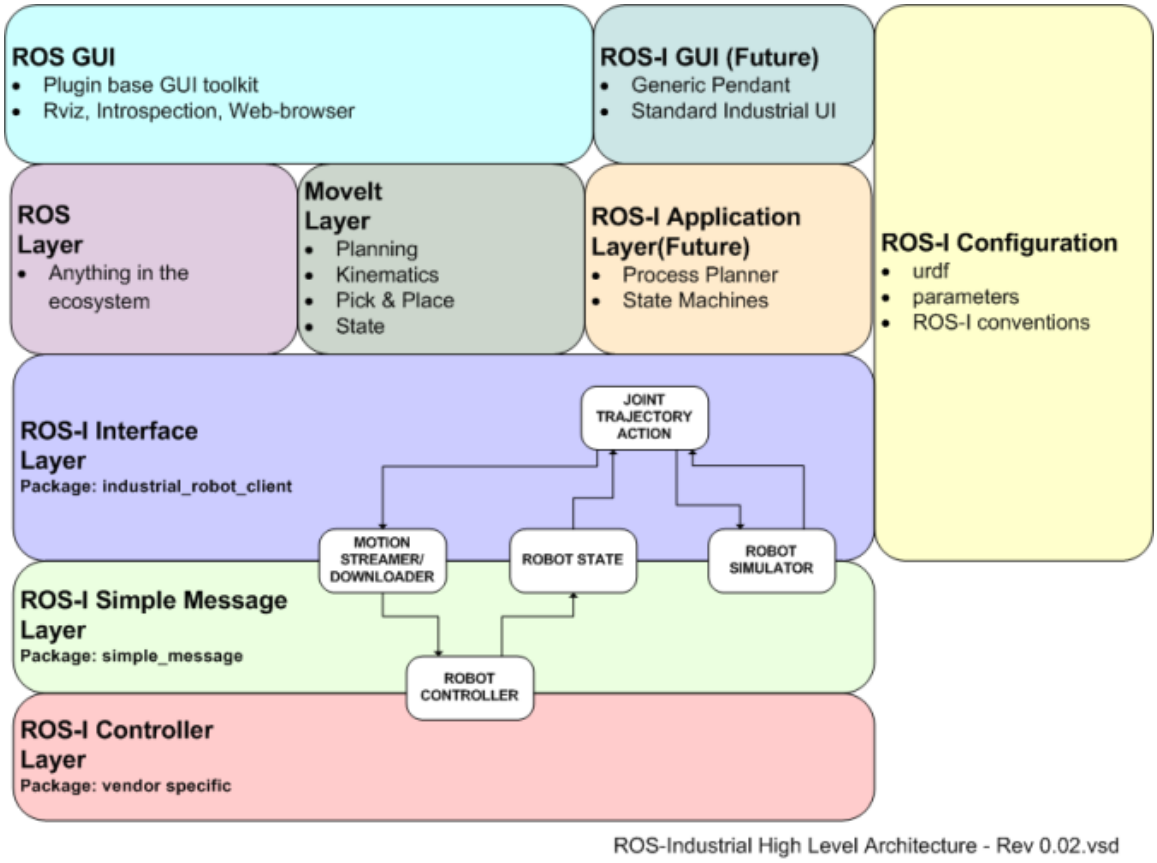


Obrázek 2.1: Princip systému ROS. Převzato z [1]

2.2 ROS-Industrial

V roce 2011 systém ROS expandoval na ROS-Industrial, aby přinesl pokročilý robotický software do oblasti průmyslové automatizace, protože i přes velký potenciál nebyl systém ROS v oblasti průmyslu doposud využíván. ROS-Industrial je stabilní spolehlivá verze ROSu, která splňuje požadavky pro průmyslové prostředí a aplikace. ROS je využíván pouze v univerzitních výzkumných centrech, kde se stále vyvíjí a zkoumá. Oproti tomu ROS-Industrial je uzpůsoben i pro průmyslové využití. ROS-Industrial spojuje dohromady všechny možné knihovny a funkcionality do jednoho celku. Následující seznam poukazuje na největší výhody a přínosy ROS-Industrial. [2]

1. Usnadňuje získávání inovací z univerzitních výzkumných center do průmyslového sektoru
2. Umožňuje uživatelům přidávat nebo měnit zdrojový kód dle vlastních potřeb
3. Nabízí open-source ovladače nezávislé na výrobci

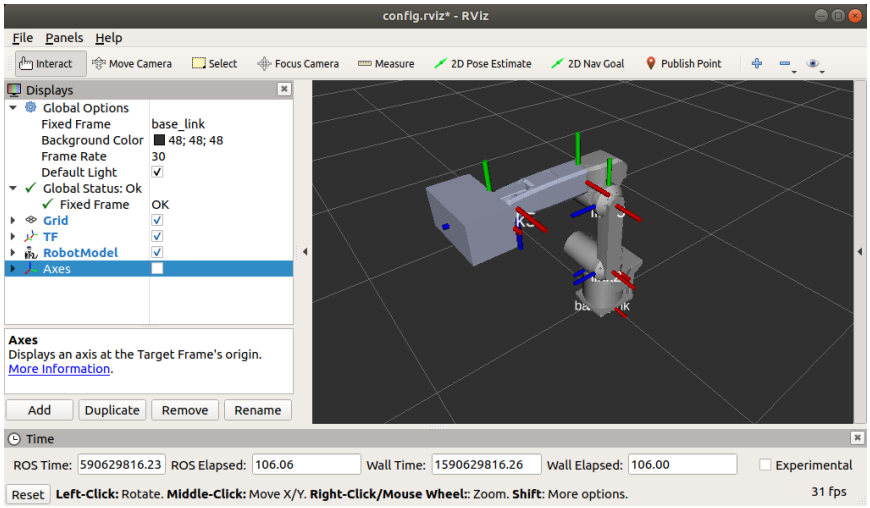


Obrázek 2.2: Architektura ROS-Industrial. Převzato z [17]

2.3 RViz – ROS Visualization

RViz je grafické uživatelské rozhraní, které vyzualizuje reálný stav robota. Dále může zaznamenávat obraz z kamer, či hodnoty snímané z čidel. Ve skutečnosti se jedná o vykreslování stavů uzlů (nodes) ROS a MoveIt. Je to uzel, který je subscriberem různých topiků,

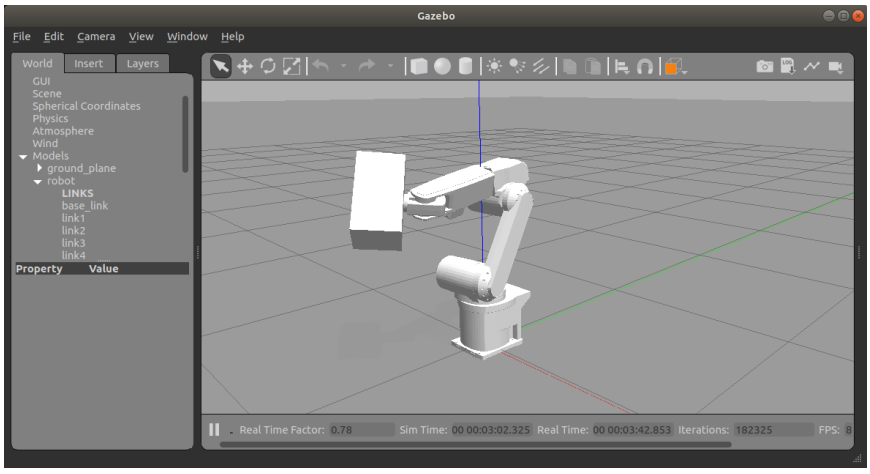
a ty přenáší zprávy vycházející z plánovacích algoritmů. Topik, pro který je *move_group* uzel publisherem a RViz od něj přijímá zprávy, je */move_group/display_planned_path*. Ve zprávách tohoto topiku se udává nejen poloha, ale také rychlost pohybu. MoveIt přináší plnohodnotnou kompatibilitu s RViz, díky tomu je možné ve vizualizaci nastavit robotickému ramenu polohu, do které by se měl robot přesunout. MoveIt následně naplánuje trajektorii ramene, kterou provede buď v simulaci, nebo na reálném robotu.



Obrázek 2.3: Ukázka prostředí RViz

2.4 Gazebo

Gazebo[1] je simulační nástroj pro simulaci robota a prostředí. Při integraci s ROSem dokáže efektivně komunikovat pomocí zpráv a podávat zpětné vazby jako reálný robot. Obdobně jako MoveIt, dokáže Gazebo nadefinovat model robota pomocí URDF a tím simulovat co nejrealističtější pohyb.



Obrázek 2.4: Simulační prostředí Gazebo

[1] Gazebo, `http://gazebosim.org/`

2.5 Robotické rameno

Robotické rameno je složené z příslušného počtu servomotorů a segmentů, které jsou popsány v podkapitole 2.5. Na poslední segment ramene je ve většině případů možno připevnit nějaký nástroj, či rozšiřující zařízení pro další možné rozšíření funkcionality robotického ramene. To mohou být např. kamery, senzory či gripper. Tato zařízení či nástroje budeme nazývat koncovým efektorem. Robotická ramena jsou využívána v různých průmyslech, kde čím dál tím více nahrazují člověka s monotónní prací (nejčastěji přesun tělesa z bodu A do bodu B). Robotické rameno se potom skládá z následujících částí:

- Základna (base) – první segment, který je sám o sobě nehybný a je pevně ukotvený k zemi
- Kloub (joint) – servomotor pohybující se zbylou částí ramene kolem jedné osy
- Segment (link) – část ramene mezi dvěma klouby
- Koncový efektor (end effector) – zařízení či nástroj připevněný na konci posledního segmentu

2.6 Real time

Běžný způsob, jak definovat reálný čas jako podstatné jméno, je čas, během kterého proces probíhá. V případě přídavného jména, real-time se týká počítačových aplikací nebo procesů, které mohou reagovat s nízkým zpožděním na požadavky uživatelů. Jedním z nejrozsáhlejších a nejpřesnějších způsobů, jak definovat reálný čas pro výpočetní systémy, je objasnit si, co se rozumí správným chováním v reálném čase. Systém fungující v reálném čase musí produkovat funkčně (algoritmicky a matematicky) správnou výstupní odpověď před dobře definovaným termínem vzhledem k žádosti o službu. [19]

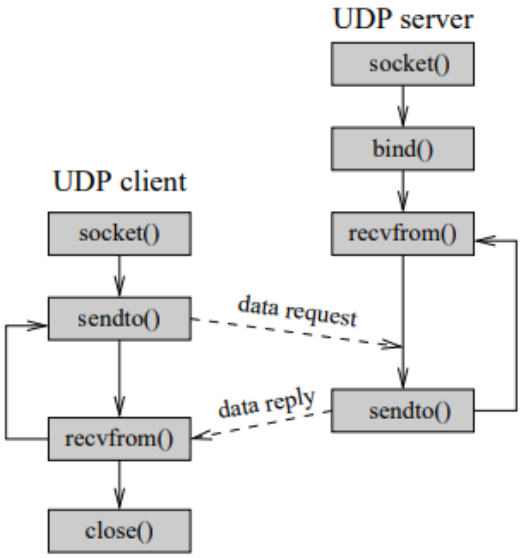
Meta-operační systém ROS pracuje na systému Linux, jež v základě v reálném čase nefunguje. To je dáno plánovací politikou, tedy jakým způsobem se bude přidělovat čas procesoru jednotlivým procesům, které si o něj žádají. Linux má v základě ve své komerční verzi "non-real-time"plánovací politiku. Změnit plánovací politiku na "real-time"je možné v programu zavoláním funkce *sched_setscheduler* s příslušnými parametry. Dále je zapotřebí aplikovat patch RT-Preempt pro kernel. Tento patch udělá z uživatelského Linuxu plně předvídatelné jádro operačního systému (fully preemptible kernel). Konkrétní užití a jak nasadit tento patch je popsáno v článku [5].

2.7 Síťová komunikace

Robotu je třeba nějakým způsobem zaslat data, kterým rozumí a dostanou ho do žádané polohy. Toho se docílí síťovou komunikací mezi počítačem s ROS a kontrolérem Melfa robota. Aby taková síťová komunikace mohla fungovat, je zapotřebí dvou prvků - klienta a serveru. V případě této práce je kontrolér CR2B-574 v roli serveru a uzel *melfa_driver_node* je klientem. Spojení mezi těmito dvěma prvky je navázáno komunikačním protokolem UDP.

Klient vyšle datagram UDP na server pomocí funkce *sendto()*. Zavolá pouze funkci *recvfrom()*, která čeká, dokud data nepřijdou od klienta. Funkce vrací kromě dat také adresu klienta, kam může server poslat odpověď. Schéma komunikace klient-server přes UDP je na obrázku 2.5. Pokud klient explicitně nepožádá o přidělení lokálního portu (funkce bind),

jádro systému mu přidělí volný port při prvním volání funkce *sendto()* a tento port mu zůstává i nadále. [10]



Obrázek 2.5: Komunikace klient-server přes UDP. Převzato z [10]

IP adresou kontroléru CR2B-574 je 192.168.0.1 a je napsána v manuálu k tomuto kontroléru. Pro komunikaci mezi síťovým rozhraním PC a kontrolérem, je zapotřebí nastavit počítač do stejné sítě jako kontrolér.

2.8 Kinematika

Pro plánování dráhy nejen pohyblivých manipulátorů, ale právě i robotických ramen, je kinematika klíčovým aspektem pro dosažení správné funkcionality. Proto zde v teoretické části zmíním informace o této pohybové studii. Následující podkapitola byla převzata z [20].

Kinematika popisuje vztahy mezi polohou, rychlostí a zrychlením skupiny pevných těles, v tomto případě se jedná o části těl robotů nebo o celé roboty [8].

Kinematika řeší schopnosti robota pohybovat se v prostoru a tuto schopnost má na starosti plánovací algoritmus, který je odlišný pro různé typy robotů a prostředí. Pozici robota je možné určit polohou koncového efektoru kartézskými souřadnicemi x, y, z v pracovním prostoru (Work Space). Kontrolér robota však očekává informaci o natočení jednotlivých kloubů, a ty dohromady tvoří kloubní prostor (Joint Space). Pomocí kinematiky je možné tyto prostorové reprezentace převádět z jedné do druhé a naopak. Rozlišují se tedy dva druhy kinematiky:

- Přímá kinematika (přímá transformace) – z hodnot natočení jednotlivých kloubů robota $\theta_1, \theta_2\theta_3, \theta_4, \theta_5, \theta_6$ se dopočítají odpovídající souřadnice x, y, z kartézského prostoru.
- Inverzní kinematika - z hodnot souřadnic x, y, z se dopočítá odpovídající natočení kloubů $\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6$.

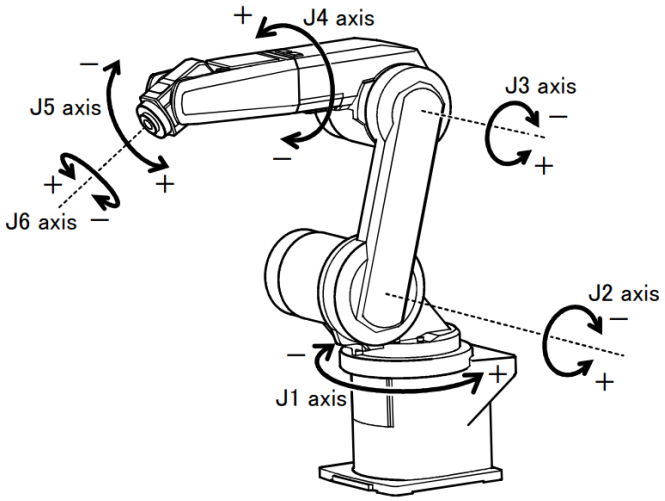
Přímá kinematika poskytuje jasné výsledky, avšak u inverzní kinematiky toto neplatí. V pracovním prostoru se dá pozice robota, dána souřadnicemi x, y, z dosáhnout více než jedním způsobem natočením kloubů v kloubním prostoru, ale také tato pozice nemusí být pro robota dosažitelná. Počet řešení je různý, závislý na počtu kloubů, omezení pohybu, vzájemné kolizi a jiných aspektech. Jaké řešení se vyhodnotí jako nejvhodnější, je poté dáno plánovačem, který by měl být vhodně zvolen pro daný typ robota.

Kapitola 3

Robotické rameno Mitsubishi Melfa RV-6SL

Konkrétní popis

Robot Melfa RV-6SL firmy Mitsubishi je angulárního (neboli úhlového) typu. Je tedy složen ze základny, kloubů a segmentů. Každý kloub se pohybuje pouze kolem jedné osy, která má omezený úhel otáčení, ten je pro každý kloub uveden ve specifikačním manuálu robota [9]. Toto rameno má šest kloubů, které se dají jednotlivě ovládat pomocí kontroléru, se kterým komunikuje počítač (Pokud tedy dále bude zmíněna komunikace s robotem, je tím myšleno přes kontrolér robota, ne se samotnou jednotkou robota).

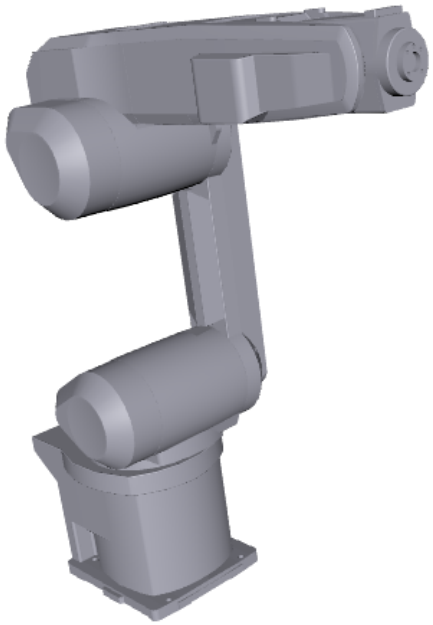


Obrázek 3.1: Model robotického ramene s popisem otáčení kloubů. Převzato z [9]

Rameno má na svém konci připojenou uchopovací hlavu (gripper) Schunk PG 70, s konstrukcí a kamerou Axis 214. Funkcionalita koncového efektoru je zanedbána, ovšem jeho existence se zanedbat nedá, proto pro výpočet setrvačnosti budou brány v úvahu pouze jeho rozměry a hmotnost.

3D model ramene

Na obrázku (3.2) je vidět 3D model ramene, který bude použit jak pro vizualizaci, tak pro definování tvaru jednotlivých segmentů po rozkladu modelu na jednotlivé segmenty.



Obrázek 3.2: 3D model ramene Mitsubishi Melfa RV-6SL [13]

3.1 Komunikace

Komunikace s robotem probíhá přes síťovou komunikaci, konkrétně protokolem UDP 2.7. Robotovi jsou v určité rychlosti posílány informace o posunu daných kloubů (je možno pohybovat více klouby zároveň) v jednotlivých paketech. To, jakou rychlostí a jaké hodnoty budou odeslány, je dáno naplánovanou trajektorií knihovnou MoveIt 4. Robot posílá zpět informaci o reálných polohách jeho kloubů.

Konfigurace

Pro možnou komunikaci s robotem je zapotřebí splnění následujících kroků:

1. Spustit v kontroléru real-time program:

```
Ovrd 10
Open "ENET:192.168.0.2" As #1
Mxt 1,1
End
```

2. Nastavit IP adresu ethernetového rozhraní na 192.168.0.2.

3. Propojit kontrolér a počítač ethernetovým kabelem.

Zdali robot opravdu naslouchá, je možné vyzkoušet pomocí příkazu *ping* na adresu robota 192.168.0.1.

3.2 Parametry

Pro správné naplánování trajektorie robota jsou potřebné přesné hodnoty jednotlivých parametrů. Nekorektní hodnoty některých parametrů by mohli, bez řádného otestování, vést k nežádoucímu chování ramene, až k jeho poškození. Dále popisuji všechny nezbytné parametry pro definici modelu URDF 3.3. Pro specifikování většiny parametrů jsem využil model ramene[1].

Tvar segmentu (link geometry)

Tvar segmentu je možný definovat dvěma způsoby. Hodnotami uvedenými v manuálu [9] (délka, šířka) nebo ze souboru formátu STL nebo DAE s 3D modelem segmentu. Aby byl popis tvaru URDF co nejpřesnější, je zapotřebí využít modelů jednotlivých segmentů. URDF podporuje pouze jednoduché geometrické tvary jako koule, krychle, kvádr. Těmito tvary by se rozměry segmentu nedaly určit tak přesně a plánování, hlavně tedy detekce kolizí, by nebyla korektní. Tvar segmentu je tedy definován tagem *<mesh>* s parametrem *filename* s cestou ke konkrétnímu modelu segmentu:

```
<mesh filename="package://rv-6sl/meshes/base_link.stl" />
```

Výpis 3.1: Definice tvaru základny ramene v popisu robota URDF

Tvar koncového efektoru je dán kvádrem o rozměrech $33x18x17$ cm. Tyto hodnoty byly naměřeny na robotu tak, aby pokryly rozměry gripperu Schunk PG 70 se všemi namontovanými přídavky jako např. držáky, kamera. Tvar gripperu tedy není definován tagem *<mesh>*, jako je u segmentů ramene, ale geometrickým tvarem *<box>* o výše uvedených rozměrech.

```
<box size="0.33 0.17 0.18" />
```

Výpis 3.2: Definice tvaru koncového efektoru

Hmotnost segmentu (link mass)

Navzdory tomu, že hmotnosti jednotlivých segmentů nejsou v manuálu k ramenu sepsány, je možné vypočítat přibližnou hmotnost segmentu poměrem daného objemu segmentu a celkové hmotnosti ramene bez hmotností servomotorů. Další možností by bylo převážit jednotlivé segmenty robota, což by bylo možné u robota vyrobeného doma na koleni. Melfa je průmyslový robot, u kterého ani jedna z již uvedených možností určení hmotnosti segmentu není možná. Hmotnost je tedy vypočtena na základě sestavy modelu robota .stp pomocí URDF exporteru[2]. Hmotnost se udává v tagu *mass*:

```
<mass value="8.2421" />
```

Výpis 3.3: Definice hmotnosti základny ramene v popisu robota URDF

[1] 3D Model Mitsubishi Melfa RV-6SL, `https://grabcad.com/library/mitsubishi-rv-6sl-1`
[2] SolidWorks to URDF Exporter, `https://github.com/ros/solidworks_urdf_exporter/releases`

Hmotnost koncového efektoru Schunk PG 70 jsem získal z oficiálních stránek tohoto produktu[3]. Je uvedena jako 1,4 kg, k tomu jsem přidal odhadnutou hodnotu držáků a kamerky 0,3 kg. Výsledná hmotnost koncového efektoru je tedy 1,7 kg.

Setrvačnost (inertial)

Setrvačnost je dána maticí setrvačnosti. Následující postup k získání matice setrvačnosti byl převzat z [6]:

Vycházejme z kinetické energie E_k, zapsané ve vektorové formě:

$$E_k = \int_m \frac{1}{2} v^2 dm \tag{3.1}$$

kde v je třísložkový vektor obvodové rychlosti, který lze rozepsat do následujících relací:

$$v = \omega \times \rho = \Omega\rho = \hat{\omega}\rho = -\hat{\rho}\omega = -\rho \times \omega \tag{3.2}$$

kde ω je vektor úhlové rychlosti, ρ polohový vektor, $\hat{\omega}$ potažmo Ω jsou antisymetrické matice, přiřazené k vektoru ω a nakonec $\hat{\rho}$ je antisymetrická matice, přiřazená k polohovému vektoru ρ:

$$\omega = \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} \tag{3.3}$$

$$\rho = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{3.4}$$

$$\hat{\omega} = \Omega = \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix} \tag{3.5}$$

$$\hat{\rho} = \begin{bmatrix} 0 & -z & y \\ z & 0 & -x \\ -y & x & 0 \end{bmatrix} \tag{3.6}$$

a dále platí relace:

$$\hat{\omega}^T \Omega^T = -\hat{\omega} = -\Omega \quad \text{and} \quad \hat{\rho}^T = -\hat{\rho} \tag{3.7}$$

před dosazením do vztahu 3.1 zbývá vyjádřit rychlost v^2:

$$v^2 = vv = v^T v \tag{3.8}$$

po dosazení do vztahu 3.1 tedy:

[3]Schunk PG 70, `https://schunk.com/no_en/clamping-technology/product/2493-0306095-pg-70/`

$$
\begin{aligned}
E_k &= \int_m \frac{1}{2} v^2 dm = \int_m \frac{1}{2} (\hat{\omega}\rho)^T (\hat{\omega}\rho) dm = \int_m \frac{1}{2} (\Omega\rho)^T (\Omega\rho) dm = \\
&= \int_m \frac{1}{2} (-\hat{\rho}\omega)^T (\hat{\rho}\omega) dm = \int_m \frac{1}{2} \omega^T \hat{\rho}^T \hat{\rho}\omega dm = \\
&= \frac{1}{2}\omega^T (\int_m \hat{\rho}^T \hat{\rho} dm)\omega = \frac{1}{2}\omega^T (\int_m -\hat{\rho}\hat{\rho} dm)\omega = \\
&= \frac{1}{2}\omega^T (\int_m -\hat{\rho}^2 dm)\omega = \frac{1}{2}\omega^T I_S \omega
\end{aligned}
\tag{3.9}
$$

Kde I_S je matice setrvačnosti, jejíž přesnou podobu je možné získat dalším roznásobením:

$$
\begin{aligned}
I_S &= -\int_m \begin{bmatrix} 0 & -z & y \\ z & 0 & -x \\ -y & x & 0 \end{bmatrix}^2 dm = \\
&= -\int_m \begin{bmatrix} 0 & -z & y \\ z & 0 & -x \\ -y & x & 0 \end{bmatrix} \begin{bmatrix} 0 & -z & y \\ z & 0 & -x \\ -y & x & 0 \end{bmatrix} dm = \\
&= \int_m \begin{bmatrix} y^2+z^2 & -xy & -xz \\ -yx & x^2+y^2 & -yz \\ -zx & -zy & x^2+y^2 \end{bmatrix} dm
\end{aligned}
\tag{3.10}
$$

$$
I_S = \begin{bmatrix} I_x & -D_{xy} & -D_{xz} \\ -D_{yx} & I_y & -D_{yz} \\ -D_{zx} & -D_{zy} & I_z \end{bmatrix}
\tag{3.11}
$$

Z výsledku je vidět, že matice setrvačnosti je symetrická. Proto nám tedy bude stačit šest hodnot, a to hodnoty z diagonály a hodnoty nad ní či pod ní. Matice setrvačnosti se dá vypočítat výše uvedeným postupem nebo využitím URDF exporter pluginu pro Solidworks, který při exportu na základě vlastností sestavy modelu tyto hodnoty vypočítá pro jednotlivé segmenty. Hodnoty jsem získal URDF exporterem a v simulaci se ukázaly býti správné. Tag *<inertia>* pro setrvačnost vypadá takto:

```
<inertia ixx="0.039989" ixy="-0.00018061" ixz="-0.0010159"
         iyy="0.040458" iyz="-0.0052802" izz="0.026957" />
```

Výpis 3.4: Definice setrvačnosti prvního segmentu ramene v popisu robota URDF

Pro hodnoty setrvačnosti koncového efektoru jsem využil matice pro kvádr, jež je definována jako:

$$
I = \begin{bmatrix} \frac{1}{12}m(h^2+d^2) & 0 & 0 \\ 0 & \frac{1}{12}m(w^2+d^2) & 0 \\ 0 & 0 & \frac{1}{12}m(w^2+h^2) \end{bmatrix}
\tag{3.12}
$$

kde: w je šířka, h výška, d hloubka, a m je hmotnost této krychle.

Po dosazení vyjde matice:

$$
I = \begin{bmatrix} 0,00868417 & 0 & 0 \\ 0 & 0,01952167 & 0 \\ 0 & 0 & 0,0200175 \end{bmatrix}
\tag{3.13}
$$

Pro určení matice setrvačnosti geometrických těles stačí pouze hodnoty z hlavní diagonály, hodnoty mimo diagonálu zůstanou nulové. Výsledný zápis XML tagu v modelu robota URDF je tedy:

```
<inertia ixx="0.00868417" ixy="0" ixz="0"
         iyy="0.01952167" iyz="0" izz="0.0200175" />
```

Výpis 3.5: Definice setrvačnosti koncového efektoru.

Limit otáčení kloubu (joint rotation limit)

Každý kloub je jinak omezený ve svém pohybu a maximální rozsah natočení jednotlivých kloubů je vypsán v manuálu [9]. Hodnoty jsou v manuálu uvedeny ve stupních, proto je potřeba hodnoty převést do radiánů vzorcem:

$$\alpha_{(radians)} = \alpha_{(degrees)} \frac{\pi}{180} \tag{3.14}$$

Pozn. Limity kloubu nemusí být pro obě strany směru pohybu stejné. Při využití URDF exporteru se limity kloubů zadávají na konci procesu, stejně jako rychlost a kroutivý moment konkrétních kloubů. K vidění na obrázku 3.4. Ve výpisu 3.6 v tagu *<limit>* je vidět zadaný maximální a minimální rozsah natočení čtvrtého kloubu.

Rychlost pohybu kloubu (joint velocity)

Hodnoty pro maximální rychlosti pohybu kloubů jsou taktéž uvedeny v manuálu [9]. Ovšem je zde také zapotřebí převést jednotky ze $^\circ/s$ na rad/s. Platí že:

$$1\,^\circ/\mathrm{s} = 0.017453\,\mathrm{rad/s} \tag{3.15}$$

proto:

$$\begin{aligned} 352\,^\circ/\mathrm{s} &= 352 \times 0,017453\,\mathrm{rad/s} = \\ &= 6,143456\,\mathrm{rad/s} \end{aligned} \tag{3.16}$$

Výsledná hodnota $6,143456\,\mathrm{rad/s}$ je maximální rychlost pohybu čtvrtého kloubu ramene. 3.6

Kroutivý moment (joint effort)

Jelikož v URDF modelu 3.3 může joint znamenat jak posuvný, tak otočný „kloub“, tak je zvolen univerzální anglický název pro kroutivý moment - effort. Kroutivý moment je v manuálu [9] uveden pouze pro poslední tři klouby. Pro první tři jsem zvolil hodnotu vyšší, chování robota bylo ověřeno v simulaci 2.4.

```
<limit lower="-2.7925" upper="2.7925"
       effort="12" velocity="6,143456" />
```

Výpis 3.6: Definice limit čtvrtého kloubu ramene v popisu robota URDF

3.3 URDF - Unified Robot Description Format

Kvalitní model robota s jeho přesnými parametry je nezbytným článkem pro správný chod plánování, detekci kolizí. Model je popsán ve značkovacím jazyce XML s využitím xacro maker. URDF se skládá z popisů kloubů, segmentů a jejich parametrů 3.2. Dále popisuje hierarchii mezi souřadnými systémy reprezentované ve stromové struktuře *TF*. Pro rameno Mitshubishi Melfa RV-6SL je URDF popis dostačující, ovšem u složitějších modelů, pracujících například se zdroji světla, které nejsou přímo spjaty s robotem samotným, by popis URDF nebylo možné použít.

Struktura

Model robota je tvořen dvěma prvky - *<link>* segmenty a *joint* klouby, kde klouby se vážou na segmenty mezi kterými leží. Ty jsou obalené do elementu *<robot>*. Základní struktura URDF je popsána následovně:

```
<?xml version="1.0" encoding="utf-8" ?>
<robot name="Nazev robota"/>
    <link name="link1"/>
    <joint name="joint1" type="Typ kloubu">
        <parent link="link1"/>
        <child link="link2"/>
    </joint>
    <link name="link2"/>
    ...
</robot>
```

Výpis 3.7: Struktura popisu robota URDF [16]

Níže vysvětlím jednotlivé elementy, parametry a atributy, které se v URDF popisu robota Melfa vyskytují. Postupně pro segment, poté pro kloub.

Segment

Segment popisuje tu část robota, která leží mezi dvěma klouby. V popisu robota URDF je tato část reprezentována elementem *link*. Tento element má tři hlavní elementy s dalšími elementy a jejich atributy: [7]

```
<link name="Nazev linku">
    <inertial>
    ...
    </inertial>
    <visual>
    ...
    </visual>
    <collision>
    ...
    </collision>
</link>
```

Výpis 3.8: Struktura segmentu popisu robota URDF [7]

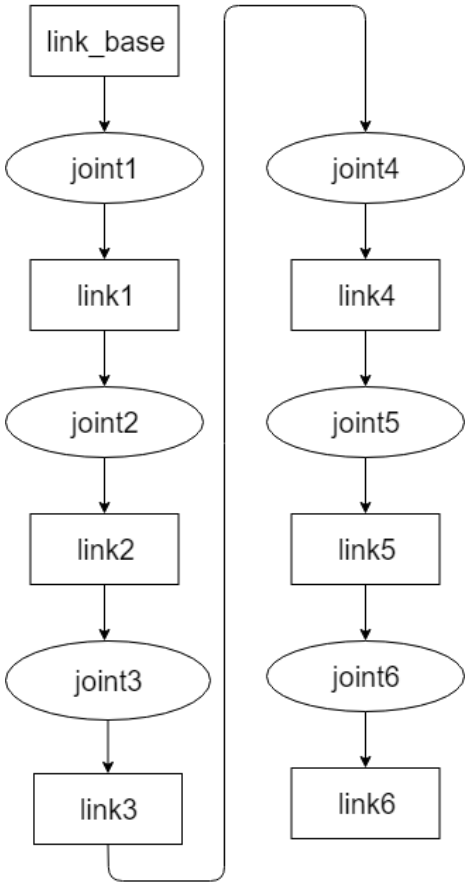
Dalšími elementy těchto hlavních elementů jsou: [7]

- **<intertial>** - obsahuje vlastnosti o setrvačnosti
  - **<origin>** - posun těžiště vzhledem k počátku segmentu
    - **xyz** - posunutí po jistých osách x, y, z
    - **rpy** - úhel natočení podle jedné z os x, y, z [rad]
  - **<mass>** - atribut **value** - hmotnost segmentu [kg]
  - **<inertia>** - hodnoty setrvačnosti viz 3.2
- **<visual>** - vlastnosti pro vizualizaci robota
  - **<origin>** - posun počátku souřadného systému vizuálního segmentu vzhledem k počátku segmentu
    - **xyz** - posunutí po jistých osách x, y, z
    - **rpy** - úhel natočení podle jedné z os x, y, z [rad]
  - **<geometry>** - tvar vizuálního segmentu viz 3.2
  - **<material>** - atribut **rgba** - umožní změnit barvu a průhlednost segmentu
- **<collision>** - vlastnosti geometrie segmentu pro detekci kolizí
  - **<origin>** - posun počátku souřadného systému vizuálního segmentu vzhledem k počátku segmentu
    - **xyz** - posunutí po jistých osách x, y, z
    - **rpy** - úhel natočení podle jedné z os x, y, z [rad]
  - **<geometry>** - tvar vizuálního segmentu viz 3.2

Kloub

Kloub spojuje dva segmenty dohromady. V popisu robota URDF je tato část reprezentována elementem *joint* s atributy *name* určující jméno kloubu a *type* jenž určuje typ kloubu. Klouby ramene Melfa jsou typu *revolute*, tedy rotační s limity. Dalšími možnými jsou např. *continuous* - rotační kloub bez limitů, nebo *prismatic* - posouvá se podél osy po stanovené limity. Elementy kloubu jsou: [4]

- **<origin>** - hodnoty posunutí (atribut **xyz**) a natočení (atribut **rpy**) počátku souřadného systému potomka vzhledem k počátku souřadného systému rodiče
- **<parent>** - název rodičovského segmentu (blíže k základně) v atributu **link**
- **<child>** - název segmentu potomka (dále od základny) v atributu **link**
- **<axis>** - osa otáčení kloubu určena vektorem **xyz**
- **<limit>** - limity pro následující atributy:
  - **lower** - udává nejmenší možný úhel natočení kloubu [m] viz 3.2
  - **upper** - nejvyšší možný úhel natočení [m] viz 3.2
  - **effort** - maximální dovolený kroutivý moment [Nm] viz 3.2
  - **velocity** - udává nejmenší možný úhel natočení kloubu [rad/s] viz 3.2



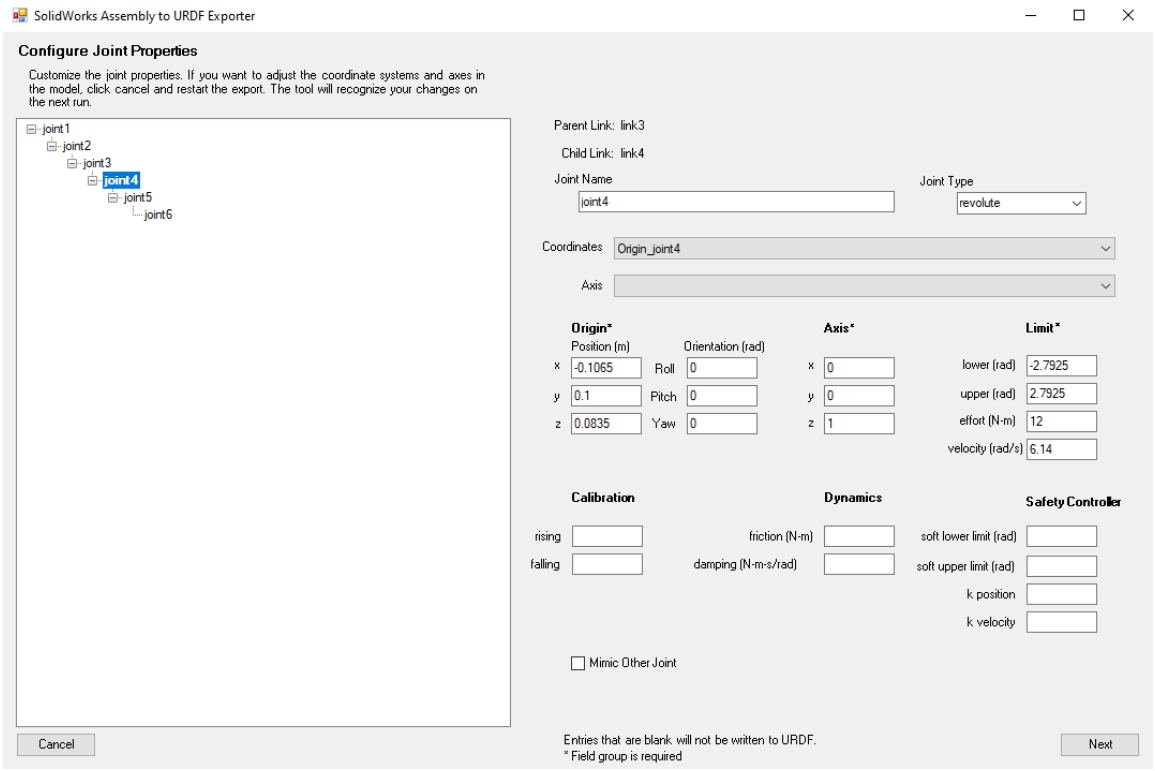
Obrázek 3.3: URDF diagram pro Mitsubishi Melfa RV-6SL

3.4 Vytvoření URDF pro Melfa RV-6SL

K vytvoření URDF a textur pro vizualizaci robota, jsem využil následující software - MeshLab[4], SolidWorks[5] s pluginem URDF exporter[6] a totožný 3D model ramene[7] s reálným ramenem.

Nainstaloval jsem plugin pro SolidWorks a importoval sestavu robota ve formátu STEP. Sestava není třeba nijak upravovat, tudíž jsem se přesunul rovnou k exporteru *Nástroje -> File -> Export as URDF*. Zde se pro každý segment vybere podsestava modelu, která mu náleží a pojmenuje se standardně podle ROS konvencí: *base_link* pro fixní segment (základnu), *linkx* pro segment a *jointx* kloub s ním pohybující joint$x$, kde x je pořadové číslo dvojice segmentu a jím náležícímu kloubu. Jedná se o robota angulárního typu, tudíž pro každý segment přidám právě jednoho potomka. Všechny segmenty jsou vybrány a vzájemně provázány viz 3.3. Pokračováním - tlačítkem *Preview and Export...* přecházím na obrazovku 3.4 pro specifikaci limitů kloubů a volby osy otáčení a dalších parametrů. *Joint Type* je typ kloubu, který je v našem případě pro všechny klouby *revolute*. To znamená, že se kloub pohybuje pouze po jediné ose a na této ose má maximální a minimální

[4]MeshLab: an Open-Source Mesh Processing Tool, `http://www.meshlab.net/`
[5]3D CAD Design Software, `https://www.solidworks.com/`
[6]SolidWorks to URDF Exporter, `https://github.com/ros/solidworks_urdf_exporter/releases`
[7]3D Model Mitsubishi Melfa RV-6SL, `https://grabcad.com/library/mitsubishi-rv-6sl-1`



Obrázek 3.4: URDF exporter v prostředí SolidWorks

limity. *Coordinates* je souřadný systém, ve kterém se zvolí osa v sekci *Axis*, podle které se kloub otáčí. Od souřadného systému se následně odvozuje pozice souřadného systému potomka. *Origin* a jeho souřadnice *(x y z)* tedy udávají posun tohoto souřadného systému vůči souřadnému systému rodiče. Tyto hodnoty jsou vypočítány automaticky na základě jednotlivých souřadných systémů každé z podsoustav. V manuálu [9] k robotu jsou vypsány maximální a minimální limity otáčení, kroutivý moment i rychlost. Je důležité dát si zde pozor na uvedené jednotky, a jsou-li ve stupních, je nutné je převést na radiány. Na další straně jsou parametry segmentů: *Origin* počátek, *Moment of Intertia* moment setrvačnosti a hmotnost, které vypočítal plugin *URDF exporter* na základě informací ze sestavy.

Tlačítkem *Export URDF and Meshes..* se vygeneruje ROS package, ve které je URDF popis robota, konfigurační soubory, launchery pro vizualizaci v RViz 2.3 a složka meshes se síťovými modely (dále jen mesh) pro každý segment. Meshe slouží jak k vizualizaci, tak ke kontrole kolizí. Přesto že URDF exporter vygeneroval meshe pro každý segment, nevygeneroval je správně. Vygeneroval je tak, že každý mesh segmentu byl vykreslený podle souřadného systému hlavní sestavy robota. Proto jsem musel v SolidWorks na šestkrát exportovat model robota (ve formátu .stl) pokaždé vykresleného podle jiného souřadného systému a v programu MeshLab dále sloučit meshe jednotlivých podsestav tvořících jeden segment se správným souřadným systémem. Ve vygenerované package (*rv-6sl*) jsem vyměnil v podsložce */meshes* meshe za nové a přesunul package na virtuální stroj s Ubuntu a ROSem.

V tomto stavu však URDF popis robota není kompletní. Vedle segmentů a kloubů je nutné popsat také koncový efektor Schunk PG 70. Ten jsem přeměřil spolu s přidělanými konstrukcemi a kamerkou, určil mu rozměry, hmotnost a dopočítal setrvačnost. Postup

k získání těchto hodnot jsem popsal v podkapitole 3.2. Segment koncového efektoru *link6* je v popisu robota URDF popsán následovně:

```
<link name="link6">
 <inertial>
     <origin xyz="2.5215E-10 -3.7993E-05 -0.0062125" rpy="0 0 0" />
     <mass value="1,7" />
     <inertia ixx="0.00868417" ixy="0" ixz="0" iyy="0.01952167"
              iyz="0" izz="0.026957" />
 </inertial>
 <visual>
     <origin xyz="0 0 0.09" rpy="0 0 0" />
     <geometry>
         <box size="0.33 0.17 0.18" />
     </geometry>
     <material name="">
         <color rgba="0.79216 0.81961 0.93333 1" />
     </material>
 </visual>
 <collision>
     <origin xyz="0 0 0.09" rpy="0 0 0" />
     <geometry>
         <box size="0.33 0.17 0.18" />
     </geometry>
 </collision>
</link>
```

Výpis 3.9: Popis koncového efektoru

Graf závislostí, po přidání koncového efektoru pro jednotlivé segmenty a klouby, je vyobrazen na obrázku 3.3.

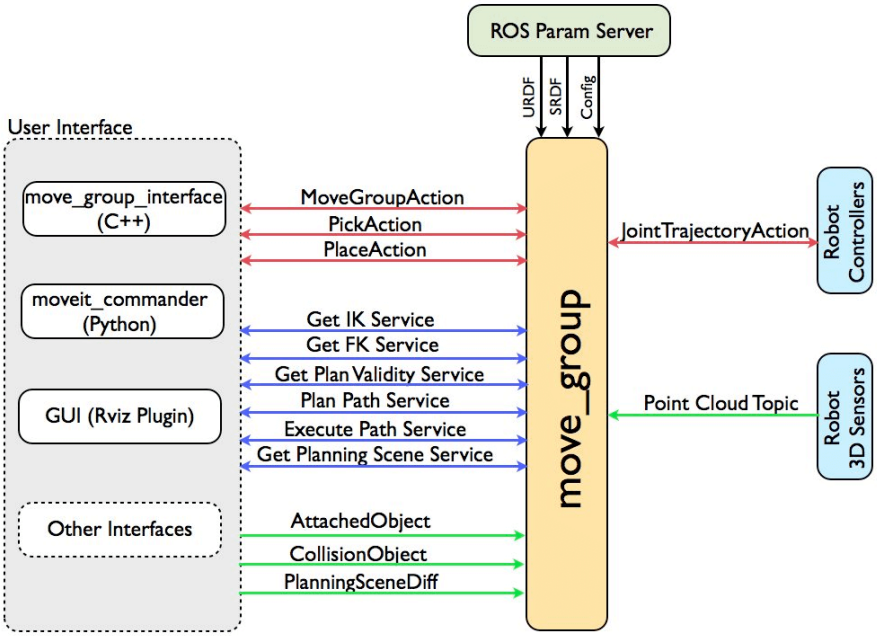
Kapitola 4

MoveIt

MoveIt [1] je plugin pro systém ROS. Zakládá si na jeho komunikaci využívá některé ROS nástroje jako například ROS vizualizér RViz 2.3 a popis robota formátem URDF 3.3. MoveIt rozšiřuje ROS o kinematiku, plánování trajektorie, kontrolu kolizí, prostorového vidění a další. V této kapitole se věnuji MoveIt architektuře a poslední podkapitolou je průvodce MoveIt Setup Assistantem pro Melfa robota.

4.1 Architektura

MoveIt Architektura je znázorněna na následujícím obrázku:



Obrázek 4.1: Architektura MoveIt. Převzato z [14]

Hlavním členem celé architektury je uzel *move_group*. Ten má za úkol naplánovat trasu pomocí vhodných plánovacích algoritmů. Aby správně fungoval, tak mu musí být poskyt-

[1] MoveIt Software, `https://moveit.ros.org/`

nuty určité parametry. Ty mu poskytuje Parameter Server, jež je součástí ROS Master uzlu. Parametry robota se tomuto serveru předají z popisu robota URDF 3.3. Semantic Robot Decription Format (SRDF) obsahuje specifikace skupin, výchozí nastavení robota a další informace. SRDF je vygenerován pomocí MoveIt Setup Assistantu 4.2. Pro provedení plánované trasy je nutná zpětná vazba robota, kterou poskytuje robot jak simulovaný, tak reálný. Zpětnou vazbu robota je třeba přečíst a publikovat hardwarovým rozhraním 5.3, tu si následně přečte JointTrajectoryAction kontrolér, který při porušení omezení, např. neplánované pozice kloubů, překročení rychlosti či nedostavení zpětné vazby, může přerušit provádění trajektorie.

Model Robota

Již z výše uvedeného schéma uzlu *move_group* můžeme vidět, že tento uzel načítá z parametrizačního serveru 3 parametry, které jsou dostačující pro reprezentaci robota. Těmito částmi jsou:

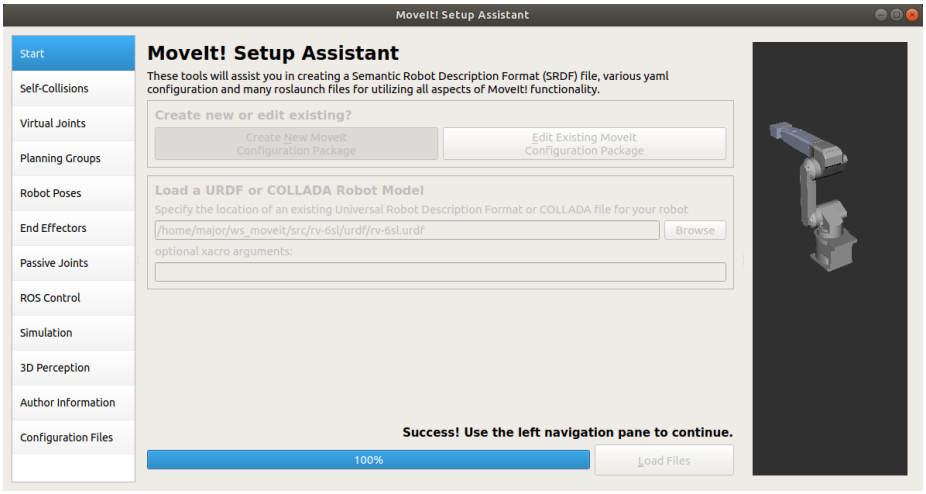
- URDF viz 3.3
- SRDF - popis, který je vytvořen při projití MoveIt Setup Assistentem 4.2. Obsahuje důležité parametry (některé z nich vytažené z modelu URDF): limity kloubů, plánovací skupiny, základní poloha robota a seznam dvojic segmentů, které se buď nikdy nemohou srazit, nebo se srazit mohou (jednoduchá matice pro zjednodušení výpočtů trajektorie)
- Config - konfigurace pro jednotlivé plánovače

Uživatelské rozhraní

Způsobem, jakým programátor komunikuje s vlastní MoveIt architekturou je přes topiky, tedy zprávami, které budeme z uživatelského rozhraní publikovat hlavnímu uzlu *move_group*. První možností, jak zprostředkovat komunikaci mezi uživatelem a MoveIt, je naprogramovat si vlastní GUI, to MoveIt umožňuje pomocí balíčku *move_group_interface* pro jazyk C++ nebo pomocí balíčku *moveit_commander* v Pythonu. Druhou z možností je využít vizualizační prostředí nabízené systémem ROS – ROS Visualization 2.3.

4.2 MoveIt Setup Assistant

Jak již název napovídá, MoveIt Setup Assistant (MSA) 4.2 je nástroj pro vytvoření balíčku, který má být základem pro plánování dráhy jakéhokoli robota pomocí knihoven MoveIt. Po splnění kroků asistentem se na základě popisu robota URDF 3.4 vygeneruje plnohodnotný balíček se specifikací robota SRDF, konfiguračními soubory pro robota dle předloženého modelu, např. pro RViz, a interakci s meta-operačním systémem ROS. Balíček se standardně pojmenovává jako název balíčku s URDF s případnými konfiguračními soubory a launchery, tedy *"název balíčku [3.4] + _moveit_config"*.



Obrázek 4.2: Úvodní strana programu MoveIt Setup Assistant s načteným modelem

4.3 Vytvoření balíčku

V této podkapitole bude popsán postup skrze MoveIt Setup Assistant, kterým jsem docílil balíčku *rv-6sl_moveit_config*.

Po nainstalování programu MSA a jeho spuštění jsem v prvním kroku **Start** vybral URDF model robota z podsložky *urdf* balíčku *rv-6sl*. MSA načetl popis robota, jenž se vyobrazil díky meshům ve stejném balíčku.

Self-Collision

Zde se vygeneruje kolizní matice s dvojicemi segmentů a označí se jejich klouby podle toho, zda se segmenty mohou dostat do kolize, či se nikdy segmenty srazit nemohou. Tato matice značně ulehčuje detekci kolize pro plánovací skupinu viz dále.

	base_link	link1	link2	link3	link4	link5	link6
base_link		☑	☐	☐	☐	☐	☐
link1	☑		☑	☑	☐	☐	☐
link2	☐	☑		☑	☐	☑	☐
link3	☐	☑	☑		☑	☑	☑
link4	☐	☐	☐	☑		☑	☐
link5	☐	☐	☑	☑	☑		☑
link6	☐	☐	☐	☑	☐	☑	

Obrázek 4.3: Matice kolizí

Bílá - značí dvojice segmentů v kolizi, **modrá** - značí vedlejší segmenty, které se nikdy nemohou srazit a **zelená** - značí ty dvojice segmentů, které se nikdy nemohou srazit, na základě omezení pohybu jednotlivých kloubů ramene.

Virtual Joints

Zde se přidávají virtuální klouby ramene, např. kloub upevňující robota k pohyblivé podstavě. V mém řešení se virutální kloub nepřidává, jelikož je již přidán do popisu robota URDF. Tento kloub se jmenuje *world_joint* je fixní a váže se na rodičovský segment *world* a potomka *base_link*. Tato dvojice se přidává do popisu robota kvůli propojení robota se světem v simulaci. Pro reálného robota je tedy tato dvojice virtuálního segmentu a kloubu zanedbatelná.

```
<link name="world"></link>
<joint name="world_joint" type="fixed">
  <parent link="world"/>
  <child link="base_link"/>
</joint>
```

Planning Groups

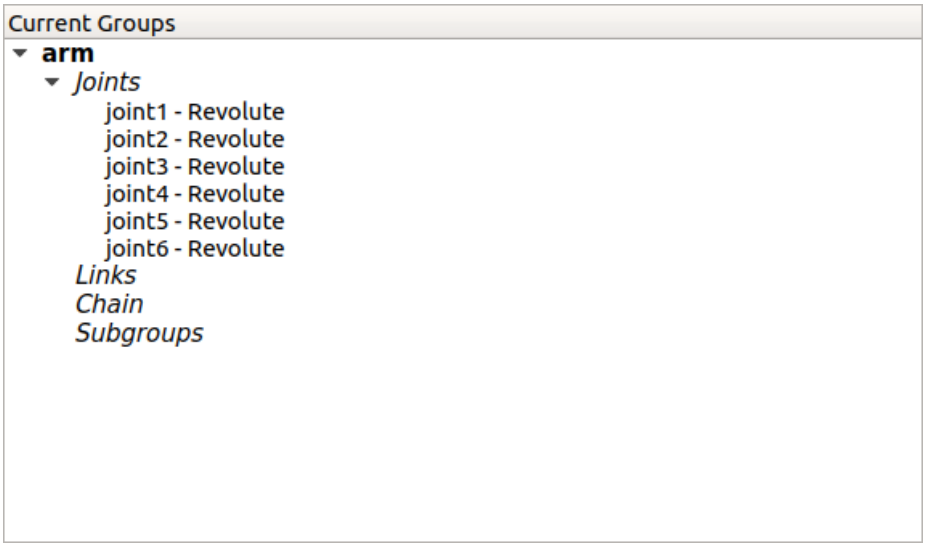
V Planning Groups je zapotřebí vytvořit skupinu kloubů, pro které bude MoveIt dráhu plánovat. V případě robota s několika rameny by se zde vytvořilo více plánovacích skupin. Avšak v případě robotického ramene Mitsubishi zde vytvořím pouze jednu plánovací skupinu. Pro průmyslové roboty se šesti stupni volnosti se zde vytváří plánovací skupiny dvě. Jedna z nich je pro plánování dráhy ramene a druhá pro koncový efektor. Funkcionalita efektoru je u ramene této práce zanedbaná, tedy jedna plánovací skupina zde postačí. Po přidání všech kloubů (bez fixního kloubu *world_joint*) do plánovací skupiny je zde ještě volba parametrů pro řešení kinematiky. V konfiguračním souboru *kinematics.yaml* 4.3 po dokončení průvodce MSA jsou tyto parametry uloženy a dají se zde později upravit.

```
arm:
  kinematics_solver: kdl_kinematics_plugin/KDLKinematicsPlugin
  kinematics_solver_search_resolution: 0.005
  kinematics_solver_timeout: 0.005
```

Význam jednotlivých parametrů:

- **kinematics_solver** - kinematický řešitel *KDLKinemticsPlugin* je poskytnut softwarem MoveIt a využívá pro výpočet Jacobiho matici
- **kinematics_solver_search_resolution** - hodnota rozlišení, které kinematický řešitel používá k prohledávání prostor. Tento parametr je tedy pro tuto práci nepodstatný, jelikož robot nevyužívá kameru ke svému pohybu
- **kinematics_solver_timeout** - maximální čas vyhrazený k vyřešení problému inverzní kinematiky od doby zahájení jistého cyklu řešení

Hodnoty přiřazené výše uvedeným záznamům jsou výchozí a ukázaly se jako funkční.



Obrázek 4.4: Plánovací skupina v prostředí MoveIt Setup Assistant

Robot Poses

Robot Poses je část, ve které je možné přednastavit, uložit a pojmenovat polohy robota. Využití se tedy dá najít v případě potřeby vytvoření sekvence poloh robota. Já jsem specifikoval základní polohu robota a několik náhodných.

End Effectors

V sekci End Effectors se upřesní plánovací skupiny ze sekce Planning Groups pro koncové efektory. Zde se speciálně zvolí plánovací skupina s dalšími parametry jako rodičovský kloub, s jehož polohou bude dále tato plánovací skupina pro efektor počítat. V tomto projektu koncový efektor nenabývá funkcionality, tudíž je tento krok přeskočen.

Passive Joints

Passive Joints jsou klouby, pro které nebude třeba publikovat jejich stav. V případě robota Mitsubishi Melfa je třeba jeho kontroléru posílat stavy všech kloubů, proto ani v tomto kroku se neprovedou žádné změny.

ROS Control

Záložka pro vytvoření konfiguračního souboru *ros_controllers.yaml* pro programové kontroléry, které zajistí komunikaci MoveIt s ROS Control. Po dokončení průvodce je tento soubor ve složce *config* s dalšími konfiguračními soubory. V tomto souboru se nachází parametry s hodnotami, určené pro komunikaci s robotem bez vlastního fyzického kontroléru. Tím jsou myšlení roboti např. vytvoření ručně se segmenty z 3D tiskárny, levnými čínskými servomotory a ty napojené na "řídící jednotku"Arduino. To neplatí pro průmyslového robota Mitsubishi Melfa. Tento konfigurační soubor popisuje následující kontroléry:

- **joint_state_controller** - kontrolér (nebo také uzel - ROS node), publikující informace o jednotlivých kloubech reálného robota topiku **/joint_states**. Tedy hlavní

uzel bude tento topik odebírat a MoveIt uzel *move_group* bude mít přístup k informacím o stavech jednotlivých kloubů, například pro vizualizaci robota v prostředí RViz.

```
joint_state_controller:
    type: joint_state_controller/JointStateController
    publish_rate: 50
```

- **joint_position_controller** - jsou kontroléry, pro jednotlivé klouby ramene jež zajišťují kontrolu a poskytnutí stavu tohoto kloubu pro *joint_state_controller*. Níže je vidět definice kontroléru pro *joint1*, tedy kloub mezi *base_link* a *link1*. Tento popis je pro každý kloub ramene od *joint1* po *joint6*.

```
joint1_position_controller:
    type: position_controllers/JointPositionController
    publish_rate: 50
    base_frame_id: base_link
    joint : joint1
```

  Popis jednotlivých parametrů kontroléru:

  - **type** - typ kontroléru udávající, jakým způsobem se bude kloub robota řídit. *JointPositionController* využívá k ovládání robota hodnoty natočení kloubů (jejich pozici)
  - **publish_rate** - rychlost publikování změn stavů kloubů hardwarovému rozhraní v [Hz]
  - **base_frame_id** - název základního segmentu, v případě ramene Mitsubishi *base_link*
  - **joint** - název kloubu, pro který bude tento kontrolér pracovat

- **joint_trajectory_controller** - následující konfigurace popisuje poslední kontrolér pro provádění trajektorií v kloubním prostoru na skupině kloubů. Trajektorie jsou specifikovány jako soubor trasových bodů, které mají být dosaženy v konkrétních časových okamžicích, které se ovladač pokouší provést. Jednotlivé body trajektorie se skládají z pozic kloubů a volitelně z rychlostí či zrychlení.

```
joint_trajectory_controller:
    type: position_controllers/JointTrajectoryController
    joints:
      - joint1
      ...
      - joint6
    constraints:
      goal_time: 2.0
      stopped_velocity_tolerance: 0
      joint1:
```

```
    trajectory: 0
    goal: 0.2
  ...
    ...
    ...
state_publish_rate:  50
action_monitor_rate: 20
```

Popis jednotlivých parametrů kontroléru:

- **type** - typ kontroléru
- **joints** - seznam kloubů, které se mají ovládat
- **constraints/goal_time** - maximální povolené zpoždění pro kontrolér, pokud je přesaženo, provádění se přeruší
- **constraints/stopped_velocity_tolerance** - maximální rychlost na konci trajektorie, po jejíž dosažení se trajektorie vyhodnotí jako úspěšná
- **constraints/<joint>/trajectory** - tolerance polohy pro konkrétní kloub. Pokud se poloha kloubu dostane mimo toleranci, trajektorie se přeruší
- **constraints/<joint>/goal** - maximální konečná chyba pozice kloubu
- **state_publish_rate** - Rychlost publikování stavu kontroléru v [HZ]
- **action_monitor_rate** - Rychlost monitorování stavu cíle

Tento konfigurační soubor pro kontroléry byl převzat jako součást již zmíněného balíčku *melfa_driver*.

Simulation

V této části průvodce vygeneruje změny v popisu robota URDF, které zajišťují komunikaci mezi ROS Control, MoveIt a simulačním programem Gazebo 2.4.

Těmito změnami je přidání elementu *transmission* pro každý kloub ramene a *gazebo* k využití Gazebo pluginu. Element *transmission* popisuje vztah mezi kloubem a akčním členem tohoto kloubu. Robot Mitsubishi Melfa se neřídí přímo přes řídící jednotku, ale přes fyzický kontrolér, tudíž element je pouze pro simulaci.

```
<transmission name="trans_joint1">
    <type>transmission_interface/SimpleTransmission</type>
    <joint name="joint1">
        <hardwareInterface>
        hardware_interface/PositionJointInterface
        </hardwareInterface>
    </joint>
    <actuator name="joint1_motor">
        <mechanicalReduction>1</mechanicalReduction>
    </actuator>
</transmission>
```

Popis parametrů elementu *transmission*:

- **type** - typ přenosu
- **<joint>** - název kloubu
- **<joint>/hardwareInterface** - specifikace přenosu pro konkrétní kloub
- **actuator** - akční člen kloubu, na který je *transmission* napojen
- **actuator/mechanicalReduction** - určuje mechanickou redukci při přenosu mezi kloubem a akčním členem

Hodnoty těchto parametrů jsou výchozí pro korektní komunikaci se simulačním prostředím Gazebo.

3D Perception

Záložka pro konfiguraci 3D senzorů, umožňujících robota vnímat 3D prostor.

Author Infromation

Informace o autorovi této MoveIt konfigurace.

Configuration Files

Poslední strana průvodce, kde je možné si navolit různé konfigurační soubory pro komunikaci robota s MoveIt. Nechal jsem tyto soubory vygenerovat všechny. Jejich úpravy a využití je popsáno dále. Nový konfigurační balíček MoveIt jsem pojmenoval standardně *rv-6sl_moveit_config* a nechal ho vygenerovat tlačítkem *Generate Package*.

Vytvořený balíček umístím do stejného ROS workspace, jako balíček s URDF. Zda-li MoveIt správně plánuje a model robota se ve vizualizaci hýbe podle očekávání, lze vyzkoušet následujícími kroky:

- Přeložit balíček v pracovním prostoru ROS. Tedy ve *ws_moveit/src* spustit příkaz:

  ```
  catkin_make
  ```

- Spustit demo příkazem:

  ```
  roslaunch rv-6sl_moveit_config demo.launch use_gui:=true
  ```

  Tímto se spustí vizualizační prostředí Rviz s pluginem MoveIt, kde je možné vyzkoušet správnost modelu, otáčení jednotlivých kloubů a výsledné plánování. Jedná se zde pouze o vizualizaci s falešnou zpětnou vazbou o posunu jednotlivých kloubů. Argumentem *use_gui:=true* se spustí grafické rozhraní pro uzel */joint_state_publisher*, který publikuje nastavený stav konkrétního kloubu topiku */joint_states*. Díky tomuto grafickému rozhraní je tedy možné posouvat jednotlivými klouby, ověřit si správnost jejich pohybu a rozsah.

```
major@major-pc:~$ rostopic info /joint_states
Type: sensor_msgs/JointState

Publishers:
 * /joint_state_publisher (http://major-pc:46463/)

Subscribers:
 * /robot_state_publisher (http://major-pc:42463/)
 * /move_group (http://major-pc:42429/)
```

Obrázek 4.5: Informace o propojení topiku */joint_states* při spuštění dema

O výše zmíněném topiku jsem již psal v popisu kontrolérů při vytváření MoveIt balíčku 4.3. Zde však tento topik odebírá informace o stavů kloubů z uzlu tohoto GUI, nikoliv z programového kontroléru */joint_state_controller* pro komunikaci s reálným/simulovaným robotem. S kterými uzly daný topik komunikuje (tedy který uzel je tohoto topiku subscriber či publisher), se dá zjistit příslušnými příkazy viz příloha.

4.4 Spouštěč pro MoveIt

ROS a taktéž MoveIt uvádí do provozu uzly a topiky spouštěcím souborem launcher. Ten je popsán značkovacím jazykem XML. Všechny uzly se však nespouští z jednoho jediného launcheru. MSA vygeneroval launchery s parametry speciálně pro každý uzel, které se volají z hlavního spouštěcího souboru. Například *demo.launch* 4.3 je také hlavní launcher, pouze využívající falešnou zpětnou vazbu a nespouští kontroléry pro reálného robota. Proto se tedy hodil jako šablona pro vytvoření spouštěcího souboru. Vytvořil jsem tedy spouštěcí soubor *moveit_execution.launch*.

```
<launch>
  <arg name="db" default="false" />
  <arg name="db_path"
       default="$(find rv-6sl_moveit_config)/default_warehouse_mongo_db" />
  <arg name="debug" default="false" />

  <include
    file="$(find rv-6sl_moveit_config)/launch/planning_context.launch">
    <arg name="load_robot_description" value="false"/>
  </include>

  <include file="$(find rv-6sl_moveit_config)/launch/move_group.launch">
    <arg name="allow_trajectory_execution" value="true"/>
    <arg name="fake_execution" value="false"/>
    <arg name="info" value="true"/>
    <arg name="debug" value="$(arg debug)"/>
  </include>

  <include file="$(find rv-6sl_moveit_config)/launch/moveit_rviz.launch">
    <arg name="rviz_config"
         value="$(find rv-6sl_moveit_config)/launch/moveit.rviz"/>
    <arg name="debug" value="$(arg debug)"/>
```

```
  </include>

  <include if="$(arg db)"
   file="$(find rv-6sl_moveit_config)/launch/default_warehouse_db.launch">
    <arg name="moveit_warehouse_database_path" value="$(arg db_path)"/>
  </include>

</launch>
```

Postupně jsou rozebrány některé důležité části:

- **planning_context.launch** - načítá SRDF, limity kloubů ze souboru *joint_limits.yaml* a konfiguraci kinematiky viz 4.3
- **move_group.launch** - načítá URDF, *planning_pipeline* zajišťující konexi mezi výše nahranými parametry a hlavním uzlem a načítá samotný hlavní uzel *move_group*. Dále volá *trajectory_execution.launch.xml*, který uvede do chodu programové kontroléry viz 4.3
- **moveit_rviz.launch** - spustí uzel s vizualizačním prostředím RViz a načte jeho konfigurační soubor
- **default_warehouse_db.launch** - nastartuje databázi MongoDB[2]

4.5 Rozhraní MoveIt

Způsoby, jakými je možné přistupovat k hlavnímu uzlu MoveIt *move_group*, jsou znázorněny na obrázku 4.1. Balíček *rv-6sl_moveit_config* obsahuje spouštěč 4.4 pro MoveIt GUI, jenž je první ze způsobů, jak robota ovládat. Dále je možnost vytvořit vlastní rozhraní s využitím funkcí knihovny MoveIt. Vytvořil jsem balíček *melfa_interface* s rozhraním, které zvládne několik základních funkcí jako např. vypsat aktuální polohy kloubů či přesunout robota do polohy dané natočením jeho kloubů načtených ze souboru. Využití tohoto rozhraní je popsáno v příloze A.

[2]The database for modern applications, `https://www.mongodb.com/`

Kapitola 5

Ovladač pro roboty Melfa

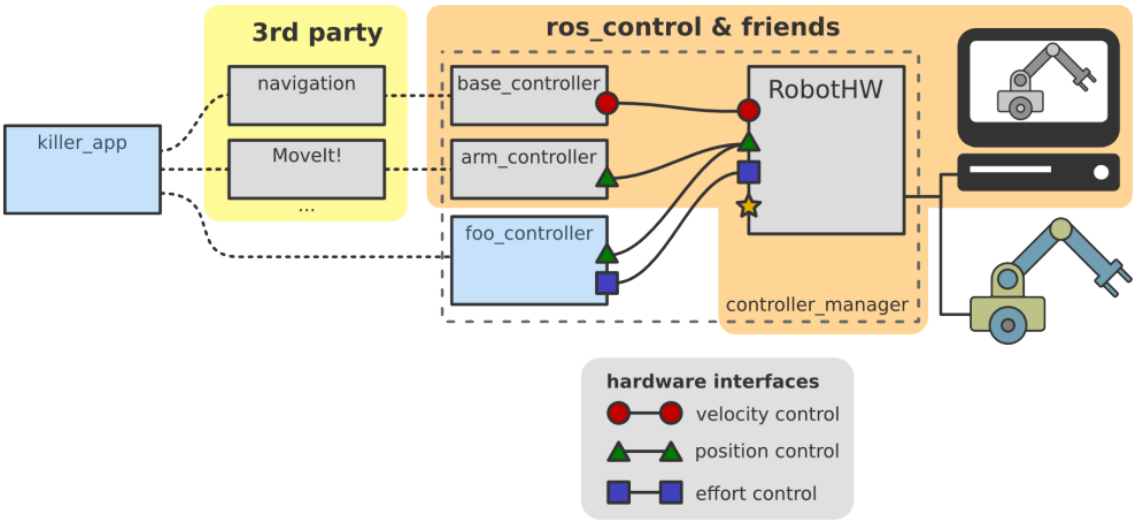
Melfa RV-6SL je průmyslový robot od Mitsubishi s vlastními servomotory, jež řídí vlastní řídící jednotka, a té posílá příkazy kontrolér taktéž firmy Mitshubishi. Není tedy možné využít ovladačů s hardwarovými rozhraními připravených od ROS, které jsou připravené pro řízení servomotorů či řídících jednotek značky Dynamixel či EsmaCAT. Využil jsem ovladače *melfa_driver*[1], který zvládne zajistit komunikaci s kontroléry, jenž komunikují s Melfa roboty. Tento ovladač zvládne na Linuxovém systému s RT-Preempt patchem řešit problémy ještě s menší odezvou viz podkapitola 2.6.

5.1 ROS Control

Než se pustím do popisování hardwarového rozhraní, bylo by dobré vysvětlit, jak takové rozhraní funguje a čeho je součástí. Hardwarové rozhraní je součástí balíčku *ros_control*. Tento balíček je součástí systému ROS. Obsahuje *controller_manager* uzel pro správu programových kontrolérů a jejich komunikaci s dalšími balíčky jako např. MoveIt a RViz ze strany uživatele a interakci s hardwarovým rozhraním ze strany robota. ROS Control, jak už jsem zmínil, zahrnuje i hardwarové rozhraní. Tato rozhraní zajišťují komunikaci s reálným robotem. ROS Control tedy zprostředkovává interakci mezi pluginy jako MoveIt, programovými kontroléry popsanými v podkapitole 4.3 a hardwarovým rozhraním. Znázorněno na obrázku 5.1.

ROS Control poskytuje některá hardwarová rozhraní již implementovaná, ty jsou však převážně pro roboty vlastní výroby. V posledních letech se ale objevují i rozhraní pro průmyslové roboty gigantů jako Kuka, Mitsubishi, Universal Robots a jiných. Tato rozhraní komunikují určitým způsobem s kontrolérem robota. V podkapitole 5.3 je využité rozhraní z balíčku *melfa_driver* vysvětleno.

[1] Melfa Driver, `https://github.com/tork-a/melfa_robot`



Obrázek 5.1: Schéma ROS Control. Převzato z [3]

5.2 Spouštěč

Obdobně jako u MoveIt kofigurace, je pro spuštění hlavního uzlu vytvořený launcher *melfa_driver/launch/melfa_driver.launch*. Jeho kód je následující:

```
<launch>
  <arg name="debug" default="false" />
  <arg unless="$(arg debug)" name="launch_prefix" value="" />
  <arg if="$(arg debug)" name="launch_prefix"
       value="gdb --ex run --args" />
  <arg name="launch_rviz" default="false"/>
  <arg name="loopback" default="false"/>
  <arg name="robot_ip" default="127.0.0.1"/>
  <arg name="realtime" default="false"/>
  <arg name="period" default="0.0071"/>

  <!-- Start MELFA driver -->
  <node name="melfa_driver" pkg="melfa_driver" type="melfa_driver_node">
    <param name="robot_ip" value="$(arg robot_ip)"/>
    <param name="realtime" value="$(arg realtime)"/>
    <param name="period" value="$(arg period)"/>
  </node>

  <!-- Start loopback node -->
  <node if="$(arg loopback)" launch-prefix="$(arg launch_prefix)"
        name="melfa_loopback" pkg="melfa_driver" type="melfa_loopback_node">
    <param name="realtime" value="$(arg realtime)"/>
    <param name="period" value="$(arg period)"/>
  </node>
```

```
  <!-- Load robot description -->
  <param name="robot_description"
         command="cat $(find rv-6sl)/urdf/rv-6sl.urdf"/>

  <!-- Load joint controller configurations -->
  <rosparam file="$(find melfa_driver)/config/controller.yaml"
           command="load"/>

  <!-- Load joint_position_controllers -->
  <node if="false" name="controller_spawner"
        pkg="controller_manager" type="spawner" respawn="false"
        args="joint_state_controller
              joint1_position_controller
              joint2_position_controller
              joint3_position_controller
              joint4_position_controller
              joint5_position_controller
              joint6_position_controller"/>

  <!-- Load joint_trajectory_controller -->
  <node name="controller_spawner" pkg="controller_manager"
        type="spawner" respawn="false"
        args="joint_state_controller
              joint_trajectory_controller"/>

  <node name="robot_state_publisher"
        pkg="robot_state_publisher" type="robot_state_publisher"/>

  <node if="$(arg launch_rviz)"
        name="rviz" pkg="rviz" type="rviz"
        args="-d $(find rv-6sl)/launch/urdf.rviz"/>

</launch>
```

Výpis 5.1: Spuštěč pro ovladač *melfa_driver*

Na začátku se specifikují argumenty a jejich výchozí hodnoty, které je možné zadat při spouštění tohoto launcheru. Argument *robot_ip* zastupuje IP adresu kontroléru, ke kterému je robotické rameno připojeno. Dalším zajímavým argumentem je *loopback*, který nastartuje uzel *melfa_loopback_node* nahrazující kontrolér robota. Nastavení argumentu *realtime* na hodnotu *true* umožní realtime komunikaci viz 2.6.

Následuje vytváření uzlů s výše uvedenými argumenty a nahrání parametrů na ROS Parametr Server. Popis robota URDF se nahraje příkazem *param*, a tím se vytvoří na serveru jeden parametr s popisem robota. Příkazem *rosparam* se načtou jednotlivé kontroléry jako parametry na základě konfiguračního souboru. Tyto kontroléry je ještě nutné uvést do provozu. O uvádění programových kontroléru do provozu se stará "spawner"uzlu *controller_manager*. Tento uzel se zároveň stará o správný chod kontrolérů. Stejně jako hlavní uzel *melfa_driver_node* je tento uzel třídou *hardware_interface::RobotHW* a je tedy schopný komunikovat s hlavním uzlem v reálném čase.

Aby bylo možné robota vidět ve vizualizaci, je zapotřebí spustit *robot_state_publisher*. Ten je subscriberem topicu */joint_states*, z kterého získává informaci o pozici jednotlivých kloubů robota. Informace zpracuje a publikuje na základě kinematického stromu modelu robota 2.8. Posledním uzlem je spuštění vizualizačního prostředí RViz, jež v našem případě není potřeba, protože RViz se spustí spouštěčem pro MoveIt 4.4.

5.3 Hardwarové rozhraní

Hardwarové rozhraní zajišťuje komunikaci s reálným robotem. Zdrojové kódy k tomuto rozhraní jsou v balíčku *melfa_driver*. Postupně zde projdu, z jakých částí se takové hardwarové rozhraní skládá. Příklady budou konkrétní z již uvedeného *melfa_driver*.

melfa_driver/include/melfa_driver/melfa_hardware_interface.h

```
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <ros/ros.h>
#include <hardware_interface/joint_command_interface.h>
#include <hardware_interface/joint_state_interface.h>
#include <hardware_interface/robot_hw.h>
#include <diagnostic_updater/diagnostic_updater.h>
#include "melfa_driver/strdef.h"
```

Include veškerých používaných knihoven. Hlavičkový soubor *melfa_driver/strdef.h* obsahuje strukturu, která bude zasílána jako zpráva robotu, jenž je pro robota srozumitelná. Struktura zasílaných dat, kterým kontrolér robotu rozumí, je poskytována dodavatelem.

```
#define JOINT_NUM (8)
```

Počet možných kloubů, které je možné ovládat. Robotické rameno Melfa RV-6SL má kloubů šest. Informace o dalších dvou kloubech jsou pro ovládání gripperu.

```
class MelfaHW:public hardware_interface::RobotHW
{
```

Třída MelfaHW popisuje vlastnosti robota. Skládá se z odesílání příkazů do motoru, řídicí smyčky a dat potřebných pro programové kontroléry. Při práci s obrazem se zde určuje způsob čtení dat z těchto zařízení.

```
public:
  MelfaHW (double period);
  void update (void);
  void write (void);
  void read (void);
  void write_first (void);
  void diagnose(diagnostic_updater::DiagnosticStatusWrapper &stat);
```

```
    inline ros::Duration getPeriod()
    {
      return ros::Duration(period_);
    }
```

Deklarace jednotlivých funkcí. Více o těchto funkcích píši v popisu .cpp souboru 5.3. Dále je tady vložená funkce *getPeriod()*, která převádí periodu robota zadanou vstupním parametrem *period* v sekundách na časovou dobu ros::Duration, s níž ROS pracuje.

```
  private:
    double period_;           //perioda zasílání zpráv
    ros::Time time_now_;      //aktuální čas
    ros::Time time_old_;      //předchozí čas - před odesláním zprávy

    std::string robot_ip_;    //IP adresa kontroléru robota
    int socket_;              //file descriptor
    struct sockaddr_in addr_; //struktura pro IP adresu

    MXTCMD send_buff_;        //buffer odesílané zprávy
    MXTCMD recv_buff_;        //buffer příchozí zprávy
    int counter_;            //čítač pro nastavení aktuální pozice robota
```

Základní proměnné pro hardwarové rozhraní s přidaným komentářem.

```
    // true if use joint7
    bool use_joint7_;
    // true if use joint8
    bool use_joint8_;
    // true if joint7 is linear joint
    bool joint7_is_linear_;
    // true if joint8 is linear joint
    bool joint8_is_linear_;
```

Proměnné pro nastavení, zda se bude využívat sedmý a osmý index pole pro ovládání gripperu a zda-li je kloub otočný nebo lineární.

```
    hardware_interface::JointStateInterface joint_state_interface;
    hardware_interface::PositionJointInterface joint_pos_interface;
```

Deklarace rozhraní pro programové kontroléry. Typ rozhraní je dán podle toho, jakým způsobem je možno robota ovládat (pozicí kloubů, rychlostí nebo točivým momentem). Tato deklarovaná rozhraní musí odpovídat programovým kontrolérům specifikovaných v 4.3.

```
    double cmd[JOINT_NUM];
    double pos[JOINT_NUM];
    double vel[JOINT_NUM];
    double eff[JOINT_NUM];
  };
```

cmd je pole s hodnotami, které se budou robotu posílat. *pos*, *vel*, *eff* jsou pole o velikosti počtu kloubů ramene, které načítají informace o stavu těchto veličin, přicházejících z kontroléru robota.

src/melfa_hardware_interface.cpp

V tomto souboru jsou implementované funkce deklarované v *melfa_hardware_interface.h* 5.3. Kód souboru *melfa_hardware_interface.cpp* je dosti obsáhlý na to, aby byl zde celý rozebrán. Je ale okomentovaný od autora, proto kód stručně popíši svými slovy a zvýrazním jen ty důležitější části HW rozhraní.

Konstruktor

Z počátku konstruktoru se inicializuje soket, načtou se hodnoty z ROS Param serveru a vyplní se struktura adresy potřebná pro odeslání paketu. Následně se vynulují buffery a vytvoří se řetězec *joint_names*:

```
std::string joint_names[JOINT_NUM] =
{
  "joint1", "joint2", "joint3", "joint4",
  "joint5", "joint6", "joint7", "joint8"
};
```

Tento řetězec obsahuje názvy kloubů shodných s těmi, které jsou uvedené v konfiguračním souboru *controller.yaml* 4.3.

```
hardware_interface::JointStateHandle state_handle (joint_names[i],
                                                   &pos[i], &vel[i],
                                                   &eff[i]);
joint_state_interface.registerHandle (state_handle);
}
registerInterface (&joint_state_interface);
```

Rozhraní pro jednotlivé kontroléry se pak musí napojit na hlavní rozhraní *joint_state_interface*, a to se následně zaregistrovat.

Následuje implementace funkcí pro přijímaní informací od kontroléru C2R-574 (*read()*) a odesílání (*write()*). Tyto funkce jsou standardní implementací s užitím funkcí knihovny BSD Sockets. Avšak před odesíláním či přijímáním paketu se plní buffer hodnotami, které čte samotný kontrolér robotu Melfa. Nejdříve budou vysvětleny jednotlivé proměnné struktury a poté hodnoty, kterými jsou v jednotlivých funkcích tyto proměnné nastaveny.

Proměnné paketu

- **Command** - Povel pro real-time komunikaci
- **SendType** - Typ dat, která se budou odesílat
- **RecvType** - Typ dat, která se budou přijímat
- **SendIOType** - Typ signálu input nebo output pro odesílaná data
- **RecvIOType** - Typ signálu input nebo output pro přijímaná data
- **BitTop** - Head bit
- **BitMask** - Označení přenosové bitové masky
- **IoData** - Nastavení toku dat na vstup či výstup
- **CCount** - Čítač provedených přenosů

write_first

Funkce *write_first* zašle robotu počáteční konfigurační zprávu. Nejprve vynuluje buffer a poté nastaví proměnné. Všechny hodnoty se nastaví na 0, až na *RecvType*. Zde se nastaví hodnota 2 oznamující robotu, že chci informaci o pozici jeho kloubů. Konstanty a struktury jsou definovány v hlavičkovém souboru se strukturou paketu *melfa_driver/include/strdef.h*.

```
send_buff_.Command = MXT_CMD_NULL;
send_buff_.SendType = MXT_TYP_NULL;
send_buff_.RecvType = MXT_TYP_JOINT;
send_buff_.SendIOType = MXT_IO_NULL;
send_buff_.RecvIOType = MXT_IO_NULL;
send_buff_.BitTop = 0;
send_buff_.BitMask = 0;
send_buff_.IoData = 0;
send_buff_.CCount = 0;
```

Nakonec se paket odešle funkcí *sendto()*.

write

Funkce *write* je obdobná jako *write_first*, naplní se struktura, odešle se paket. Ovšem odesílané hodnoty jsou už zde zajímavější. Dále je tu navíc naplnění struktury cílovými hodnotami kloubů. A pokud se má ovládat i gripper, uvádí se hodnota kloubu podle toho, zda je kloub otočný či posuvný.

```
send_buff_.Command = MXT_CMD_MOVE;
send_buff_.SendType = MXT_TYP_JOINT;
send_buff_.RecvType = MXT_TYP_JOINT;
```

Hodnota *Command* nastavena na 1, dává robotu najevo, že má něco provést. Tedy pohnout klouby, což je dáno hodnotou 2 v *SendType*. Poté robot odešle zpět aktuální natočení kloubů jako potvrzení (*RecvType*).

```
send_buff_.RecvType1 = MXT_TYP_FJOINT;
send_buff_.RecvType2 = MXT_TYP_FB_JOINT;
send_buff_.RecvType3 = MXT_TYP_FBKCUR;
```

Proměnné *RecvType1*, *RecvType2* a *RecvType3* jsou určené pro další možnosti monitorování stavu robota, tzn. robot bude místo jedné zpětné vazby posílat až tři další vazby požadovaných typů.

```
send_buff_.dat.jnt.j1 = cmd[0];
send_buff_.dat.jnt.j2 = cmd[1];
send_buff_.dat.jnt.j3 = cmd[2];
send_buff_.dat.jnt.j4 = cmd[3];
send_buff_.dat.jnt.j5 = cmd[4];
send_buff_.dat.jnt.j6 = cmd[5];
```

Zde se naplní datová struktura hodnotami v poli *cmd*. Toto pole bylo programovými kontroléry naplněno hodnotami, o které se mají jednotlivé klouby otočit, pro dosažení dalšího bodu trajektorie či cíle.

Dále jsou ve funkci již zmíněné podmínky pro hodnoty posledních dvou kloubů. Před odesláním paketu se ještě nastaví hodnota *CCount* udávající aktuální počet provedených přenosů.

read

Funkce *read* začne deklarováním proměnných - file descriptoru *fds* a časové proměnné *time*, nastavením časů *time_old_* na čas odeslání předchozí zprávy a *time_now_* na čas aktuální. Dále se vynuluje buffer, setne deskriptor a nastaví se proměnná *time* na hodnotu, po kterou bude funkce *select* čekat na uvolnění file descriptoru. Tato doba čekání je dána periodou zasílání zpráv.

```
int status = select (socket_+1, &fds, (fd_set *) NULL,
                     (fd_set *) NULL, &time);
```

Dále se zavolá funkce select, která monitoruje file descriptor a čeká na příchozí zprávu od robota. Poté se zpráva přečte funkcí *recvfrom* a uloží do bufferu *recv_buff_*. V tomto bufferu jsou teď aktuální hodnoty natočení kloubů robota, a ty se nahrají do pole *pos*. Z tohoto pole už jsou hodnoty natočení kloubů dostupné programovým kontrolérům.

Dále funkce při první komunikaci (bez prvního paketu funkce *write_first*) nastavuje aktuální pozici kloubů, jejichž hodnoty byly zjištěny prvním paketem. Nakonec se inkrementuje počet odeslaných paketů.

diagnose

Funkce pro hlídání periody. Monitoruje, zda se dodržuje pravidelné zasílání paketů.

src/melfa_driver_node.cpp

Tento soubor reprezentuje uzel, který již může být zavolán a spuštěn spouštěčem *melfa_driver/launch/melfa_driver.launch* 5.2.

```
int main (int argc, char **argv)
{
  // init ROS node
  ros::init (argc, argv, "melfa_driver");
```

V *main* funkci se nejprve inicializuje ROS node, který se bude jmenovat *melfa_driver*. Jeho první dva parametry jsou pro načtení argumentů a třetí je název uzlu. *ros::init* se musí zavolat vždy před používáním dalších ros funkcí jako např. následující deklarace *ros::NodleHandle nh*.

```
ros::NodeHandle nh;
```

NodeHandle je hlavním přístupovým bodem ke komunikaci se systémem ROS. Při vytvoření prvního handleru uzlu zavolá funkci *ros::start*, jenž nastartuje funkcionalitu uzlu. Uzel je aktivní, dokud není handler zničen, poté se zavolá funkce *ros::shutdown* a uzel se ukončí.

```
// Parameters
bool realtime;
ros::param::param<bool>("~realtime", realtime, false);
double period;
ros::param::param<double>("~period", period, 0.0071);
```

Nastavení parametrů *realtime* a *period* z ROS Param serveru.

```
// create hardware interface
MelfaHW robot(period);
```

Vytvoření instance hardwarového rozhraní popsaného výše.

```
controller_manager::ControllerManager cm (&robot, nh);
```

Uvede do chodu *controller manager* viz 5.2. Jeho parametry jsou hardwarové rozhraní a handler na tento uzel.

```
// diagnostic_updater
diagnostic_updater::Updater updater;
updater.setHardwareID("melfa_driver");
updater.add("diagnose", &robot, &MelfaHW::diagnose);
```

Deklarace a inicializace monitoringu *updater*.

```
if (realtime)
{
  struct sched_param param;
  memset(&param, 0, sizeof(param));
  int policy = SCHED_FIFO;
  param.sched_priority = sched_get_priority_max(policy);
  ROS_WARN("Setting up Realtime scheduler");
  if (sched_setscheduler(0, policy, &param) < 0) {
    ROS_ERROR("sched_setscheduler: %s", strerror(errno));
    ROS_ERROR("Please check you are using PREEMPT_RT kernel
                and set /etc/security/limits.conf");
    exit (1);
  }
  if (mlockall(MCL_CURRENT|MCL_FUTURE) < 0)
  {
    ROS_ERROR("mlockall: %s", strerror(errno));
    exit (1);
  }
}
```

Funkcionalita realtime, která zmenší odezvu tohoto rozhraní na základě změny politiky přiřazování vláken procesům *sched_setscheduler* a zamykání stránek (bloků paměti) *mlockall*.

```
// set spin rate
ros::Rate rate (1.0 / robot.getPeriod().toSec());
```

ros::Rate určuje frekvenci, jakou bude tikat hlavní smyčka 5.3. Frekvence je dána periodou specifikovanou při spouštění uzlu, výchozí $0,0071s$.

```
ros::AsyncSpinner spinner (1);
spinner.start ();
```

Zde se vytvoří a spustí spinner. Ten je potřebný pro odbavování požadavků subscriberů. Rezervuje tedy potřebný počet vláken.

```
  // write first setting packet
  robot.write_first ();
```

Odešle se první paket a začne hlavní ROS smyčka, ve které se načte odpověď robota se stavem kloubů funkcí *read*, controller manager řekne programovým kontrolérům, aby aktualizovali hodnoty kloubů v poli *cmd* a poté odešle nové hodnoty zpět robotu pomocí *write*. Před koncem smyčky se aktualizuje monitoring a smyčka se uspí, dokud neuběhne daná perioda.

```
  while (ros::ok ())
  {
    // Wait for reciving a packet
    robot.read ();
    cm.update (ros::Time::now(), robot.getPeriod());
    robot.write ();
    // Update diagnostics
    updater.update();
    rate.sleep ();
  }

  spinner.stop ();
  return 0;
}
```

ROS smyčka se ukončí tehdy, pokud už uzel není ve stavu *ros::ok*, to může být zapříčiněno vypnutím uzlu (CTRL-C), zničením všech handlerů uzlu či zavoláním *ros::shutdown*. Poté se zastaví spinner a uzel se ukončí.

Kapitola 6

Testování

Testování je rozděleno na tři části:

- Test ramene
- Test modelu
- Test ovládání

Test ramene

V úplném počátku práce bylo testováno, zda-li kontrolér ramene dokáže reagovat na zasílané zprávy. K tomu byl poskytnut real-time program, který zaslal první zprávu, přijal odpověď od robota a zaslal mu tyto informace zpět. Než zprávu odeslal, mohla být přijatá data modifikována, a tím se zaslaly robotu nové polohy kloubů a robot se na základě velikosti inkrementů poloh jednotlivých kloubů hýbal. Tím bylo ověřeno, že kontrolér robota zvládne real-time řízení a má smysl problém real-time řízení přenést do prostředí ROS.

Test URDF modelu

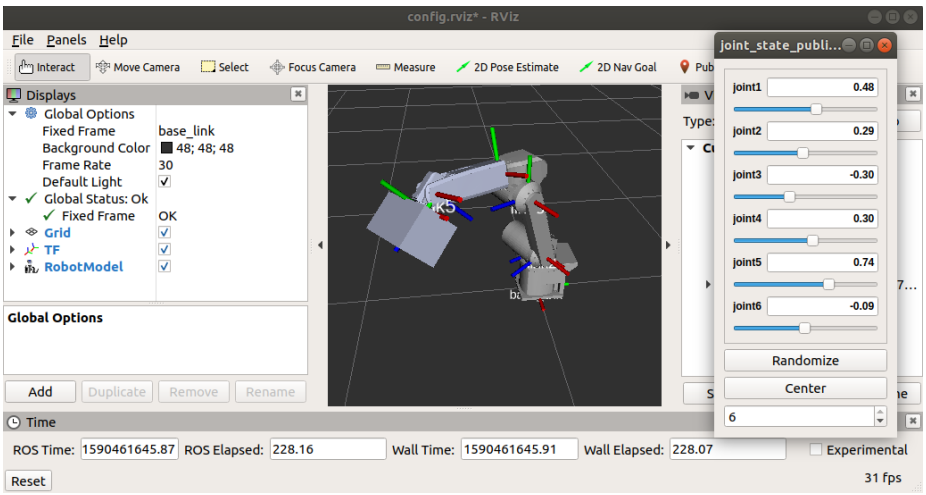
Značná část práce je věnována popisu robota URDF. Tento model musí být co nejvíce identický s reálným robotem, proto je na jeho kvalitu třeba klást velký důraz. Z počátku bylo největším problémem správně nastavit jednotlivé souřadné systémy s vizuálním a kolizním modelem tak, aby se vizualizovaný robot dokázal hýbat stejně, jako robot reálný.

Testování URDF probíhalo ve vizualizačním prostředí RViz. Po vygenerování balíčku s popisem robota URDF exporterem, se současně vytvořil spouštěč pro RViz *display.launch* pro zobrazení modelu. Tento launcher společně s *joint_state_publisher*em se spustí příkazem:

```
roslaunch rv-6sl display.launch
```

A následně se otevře prostředí RViz. Při prvním spuštění se nezobrazí *joint_state_publisher*, model robota ani struktura TF s osami souřadných systémů, jak je zobrazeno na obrázku 6.1. Pro *joint_state_publisher* je zapotřebí nastavit ve spouštěči parametr *use_gui* na *true*. Model robota, TF a další vyobrazení se přidají tlačítkem *Add*. Dále je možné upravit velikosti zobrazení os a jiné. Všechna tato nastavení je dobré uložit jako konfigurační soubor a přidat do spouštěče, jelikož by po každém spuštění RVizu spouštěčem *display.launch* byla

přidaná vyobrazení a nastavení ztracena. Tímto je prostředí pro testování modelu připraveno.



Obrázek 6.1: Testování modelu v RViz

Pomocí grafického rozhraní *joint_state_publisher*u bylo testováno, zda se jednotlivé klouby otáčí podle správných os a jsou správně nastavené limity, zda jsou vizuální a kolizní objekty vykresleny dle správného souřadného systému a další. Největším nepřítelem při modelování URDF je pro nového uživatele *<origin>*, více o tomto tagu je uvedeno v sekci segmenty 3.3.

Dosažení uspokojivého vzhledu a správného polohování kloubů ještě neznamená, že je popis robota správný. Limity kloubů jsou ověřené pouze pro otáčení, avšak pro úplnou definici robota za pomocí URDF jsou zapotřebí ještě maximální rychlost a maximální kroutivý moment každého kloubu. Tyto parametry se ověřují až v simulaci. Robot Melfa má na sobě přidělaný koncový efektor s konstrukcemi a kamerou, jež jsou zapotřebí vymodelovat a do popisu URDF přidat.

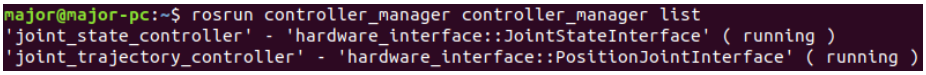
Test Ovládání

Projitím MoveIt Setup Assistantem se nevygenerovali pouze nezbytné soubory pro navázaní komunikace se systémem ROS, či sémantický popis robota SRDF, ale také spouštěč *demo.launch*, který má skvělé využití - testování knihovny MoveIt, pokud programátor stále nemá přesně definované programové kontroléry pro komunikaci se simulačním prostředím či reálným robotem. Příkazem:

```
roslaunch rv-6sl_moveit_config demo.launch
```

se spustí RViz společně s MoveIt GUI a je možné otestovat, zda plánovač pro daný model funguje. V tomto stavu je rameno pouze vizualizováno a zpětná vazba robota pro potvrzení provádění trajektorie je zajištěna uzlem *robot_state_publisher*. Natočení kloubů MoveIt publikuje na topik */joint_states*. Doposud se jednalo stále o vizualizaci.

Pro provádění trajektorie na reálném robotu je zapotřebí programových kontrolérů a hardwarového rozhraní. Programové kontroléry má na starosti *controller_manager*. Způsob jakým si ověřit, zda-li jsou kontroléry vytvořeny a jestli běží, je na následujícím obrázku.



Obrázek 6.2: Ověření běhu programových kontrolérů

Ověřování komunikace proběhlo se simulovaným robotem v prostředí Gazebo. Simulovaný robot se spustí spouštěčem *gazebo.launch* z balíčku *rv-6sl_moveit_config*. Pro simulovaného robota je možné naplánovat dráhu a následně ji provést přes MoveIt GUI. Funkce rozhraní balíčku *melfa_interface* byly testovány taktéž v simulačním prostředí Gazebo.

Testování reálného ramene bylo prováděno na začátku práce, zda-li robot dokáže takový typ komunikace přijmout a reagovat na něj. Výsledek této práce, tedy ovládání robota pomocí ROS a MoveIt, nebyl otestován na reálném robotu kvůli světové virové pandemii. Výsledek byl však testován v simulačním prostředí, jenž dokáže robotické rameno nahradit a rameno zde bylo ovládáno. Podle porovnání kontroléru *CR750-Q*, na kterém byl ovladač využit a kontrolérem CR2B-574 robota této práce, není v real-timeovém řízení rozdíl. Proto se domnívám, že výsledek této práce bude taktéž funkční na robotickém rameni Melfa RV-6SL.

Kapitola 7

Závěr

Cílem této práce bylo naplánovat dráhu robotickému ramenu Mitsubishi Melfa s využitím knihovny MoveIt. To zahrnuje zajištění korektních parametrů pro programový popis robota, zajištění modelu robota a vytvoření příslušných konfiguračních balíčků (pro komunikaci či vizualizaci), nezbytných pro dosažení daného cíle. Díky výše uvedenému bylo dosaženo pohybu ramene na základě naplánované dráhy knihovnou MoveIt s využitím jejího grafického uživatelského rozhraní. Pro interakci robota se systémem ROS jsem převzal volně dostupný ovladač melfa_driver a upravil ho pro potřeby této práce [11]. Testování funkčnosti ramene bylo prováděno v simulačním prostředí Gazebo. Správnost popisu robota URDF byla testována za pomoci vizualizačního prostředí RViz.

Pro využití úplného potenciálu robotického ramene Melfa je zapotřebí identický model koncového efektoru společně s konstrukcemi a kamerou. Poté systém ROS s dalšími volně přístupnými knihovnami umožní robotu vnímání prostoru. MoveIt tuto vlastnost plně podporuje a nevyužít jeho možností by byla škoda. V případě pokračování této práce by bylo vhodné se těmito problémy zabývat.

Tato práce mi pomohla porozumět problematice v oblasti robotiky. Obeznámil jsem se s open-source robotickým systémem, který je velmi komplexní a má toho hodně co nabídnout. Systém ROS tvoří dohromady s MoveIt velmi mocný open-source nástroj, jenž má opravdu velký potenciál nejen v průmyslu. V této práci se soustředím převážně na umožnění ovládání reálného robota prostřednictvím systému ROS společně s MoveIt. Práce obsahuje základní postupy pro zprovoznění vlastního robotického ramene a může na ni být nahlíženo jako na návod.

Literatura

[1] Achmad, M., Priyandoko, G., Roali, R. a Daud, M. Tele-Operated Mobile Robot for 3D Visual Inspection Utilizing Distributed Operating System Platform. *International Journal of Vehicle Structures and Systems*. 2017. DOI: 10.4273/ijvss.9.3.12. Dostupné z: https://www.researchgate.net/figure/Nodes-communication-model-in-the-ROS-environment-Fig-5-describes-the-proposed-ROS_fig2_319566597.

[2] Carette, J. What is ROS and ROS-Industrial for Robots? 2014. Dostupné z: https://blog.robotiq.com/bid/70845/What-is-ROS-and-ROS-Industrial-for-Robots.

[3] Chitta, S., Marder Eppstein, E., Meeussen, W., Pradeep, V., Rodríguez Tsouroukdissian, A. et al. Ros_control: A generic and simple control framework for ROS. *The Journal of Open Source Software*. 2017. DOI: 10.21105/joss.00456. Dostupné z: http://www.theoj.org/joss-papers/joss.00456/10.21105.joss.00456.pdf.

[4] URDF. *Joint element*. [cit. 2020-23-05]. Dostupné z: http://wiki.ros.org/urdf/XML/joint.

[5] Kautilya, C. How to setup PREEMPT RT on Ubuntu 18.04. 2020. Dostupné z: https://chenna.me/blog/2020/02/23/how-to-setup-preempt-rt-on-ubuntu-18-04/.

[6] Kazda, L. *Měření momentu setrvačnosti*. Praha, CZ, 2015. Bakalářská práce. České vysoké učení technické, Fakulta strojní. Dostupné z: https://dspace.cvut.cz/bitstream/handle/10467/63453/F2-BP-2015-Kazda-Lukas-Bakalarska%20prace.pdf.

[7] URDF. *Link element*. [cit. 2020-23-05]. Dostupné z: http://wiki.ros.org/urdf/XML/link.

[8] Lumelsky, V. J. *Sensing, Intelligence, Motion: How Robots and Humans Move in an Unstructured World*. Wiley-Interscience, 2005. ISBN 978-0471707400.

[9] CORPORATION, M. E. *RV-6S Series Standard Specifications Manual*. 2009.

[10] Matoušek, P. *Síťové aplikace a jejich architektura*. Vutium, 2014. ISBN 978-8021437661.

[11] Tajima, R. *Melfa Driver*. 2018 [cit. 2020-17-05]. Dostupné z: https://github.com/tork-a/melfa_robot.

[12] ROS. *Ubuntu install of ROS Melodic*. [cit. 2020-27-05]. Dostupné z: http://wiki.ros.org/melodic/Installation/Ubuntu.

[13] Barbosa, F. *3D Model Mitsubishi Melfa RV-6SL* [online]. 2014 [cit. 2020-18-01]. Dostupné z: https://grabcad.com/library/mitsubishi-rv-6sl-1.

[14] Sucan, I. A. a Chitta, S. *MoveIt Software* [online]. 2016 [cit. 2019-21-01]. Dostupné z: https://moveit.ros.org/.

[15] Dave Coleman, S. G. R. H. *Getting Started*. [cit. 2020-27-05]. Dostupné z: http://docs.ros.org/melodic/api/moveit_tutorials/html/doc/getting_started/getting_started.html.

[16] URDF. *Robot element*. [cit. 2020-23-05]. Dostupné z: http://wiki.ros.org/urdf/XML/robot.

[17] Hoorn, G. vd. *ROS Industrial*. [cit. 2020-23-05]. Dostupné z: http://wiki.ros.org/Industrial.

[18] Armin Hornung, A. B. a Adam Leeper a. *ROS - Documentation* [online]. Creative Commons Attribution 3.0, 2019 [cit. 2020-18-01]. Dostupné z: http://wiki.ros.org/Documentation.

[19] Siewert Sam, P. J. *Real-Time Embedded Components And Systems: With Linux and RTOS*. Mercury Learning and Information, 2016. ISBN 978-1942270041.

[20] Wágner, P. *Metoda plánování trajektorie robota v reálném čase*. Olomouc, CZ, 2014. Doktorská disertační práce. VŠB-TU Ostrava, Fakulta elektrotechniky a informatiky, Katedra kybernetiky a biomedicínského inženýrství. Dostupné z: https://dspace.vsb.cz/bitstream/handle/10084/106249/WAG019_FEI_P2649_2612V045_2014.pdf.

Příloha A

Použití

V této příloze je popsáno, jak použít balíčky pro ovládání robotického ramene. Pokud uživatel již pracuje se systémem ROS, může přeskočit první část Základní kroky.

Základní kroky

V prvé řadě je zapotřebí mít nainstalovaný robotický operační systém ROS Melodic LTS na Ubuntu v18.04. Pro hladkou instalaci doporučuji postup z oficiálních dokumentačních stránek ROS [12].

Dále je zapotřebí doinstalovat a nakonfigurovat překladový systémem ROS pro vytvoření pracovního prostředí *workspace* společně s knihovnou MoveIt stejně jako při instalaci systému ROS odkáži na dokumentaci k MoveIt, konkrétní postup je dostupný na [15].

Do vytvořeného pracovního prostředí *ws_moveit/src* se vloží balíčky, jež jsou výsledkem této práce. Poté se balíčky musí sestavit z adresáře pracovního prostředí *ws_moveit* příkazem:

```
catkin_make
```

Po sestavení je možné balíčky plně využít.

Příprava simulace

Pro simulování robota je zapotřebí simulační prostředí Gazebo verze 9 a vyšší. Ta by měla být nainstalována společně s instalací systému ROS po absolvování výše zmíněného postupu [12]. Simulace robota se spustí příkazem:

```
roslaunch rv-6sl_moveit_config gazebo.launch
```

Příprava reálného ramene

Aby bylo reálné rameno připraveno na komunikaci s ROS PC je zapotřebí splnění následujících kroků:

1. Spustit v kontroléru real-time program:

```
Ovrd 10
Open "ENET:192.168.0.2" As #1
Mxt 1,1
End
```

2. Nastavit IP adresu ethernetového rozhraní počítače s ROS na 192.168.0.2.
3. Propojit kontrolér a počítač ethernetovým kabelem.
4. Při využívání virtuálního stroje nastavit typ síťového rozhraní na Síťový most

Po zajištění výše uvedených bodů je potřeba otevřít terminál a spustit ovladač příkazem:

```
roslaunch melfa_driver melfa_driver.launch
```

Ovládání

Ovládání robota, ať už v simulaci nebo reálného, je možné několika způsoby. Jakými, je popsáno dále.

MoveIt GUI

První z možností je ovládat robota pomocí MoveIt GUI. Zde se určí poloha robota, plánovač naplánuje trajektorii, a tu následně provede. Grafické rozhraní MoveIt se spustí příkazem:

```
roslaunch rv-6sl_moveit_config moveit_execution.launch
```

MoveIt Rozhraní

K využití tohoto rozhraní je zapotřebí spuštěný výše zmíněné MoveIt GUI. V dalším terminálu lze pak využít rozhraní následujícím příkazem:

```
rosrun melfa_interface run <x> <file>
```

kde:

- x - celočíselná hodnota - volba funkce
- file - absolutní cesta k souboru, pro volby 1, 2, 3

Na základě volby funkce je soubor využit pro čtení či zápis. Formát načítaných či vypisovaných dat je následující:

```
j1 j2 j3 j4 j5 j6
```

Tedy úhel v radiánech pro každý kloub od začátku souboru oddělených mezerami. Příklad volání:

```
rosrun melfa_interface run 2 "/home/major/ws_moveit/src/in.txt"
```

Možné funkce

- **0** - uvede robota do výchozí polohy
- **1** - vypíše do souboru informaci o aktuálním natočení kloubů robota
- **2** - načte ze souboru, o kolik se má jaký kloub ramene pohnout a pohyb provede
- **3** - načte ze souboru cílovou polohu robota, danou natočením jednotlivých kloubů a tu provede

GUI pro klouby

Pro ovládání jednotlivých kloubů s nastavením rychlosti je možné využít grafického rozhraní pro *joint_trajectory_controller*. Ten se spustí příkazem:

```
rqt -s rqt_joint_trajectory_controller/JointTrajectoryController
```