VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

REAL-TIME ŘÍZENÍ ROBOTA MELFA POMOCÍ ROS

REAL TIME MELFA CONTROLLING USING ROS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **JAKUB LIŠKA**
AUTHOR

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

BRNO 2021

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav inteligentních systémů (UITS) Akademický rok 2020/2021

Zadání bakalářské práce

Student: **Liška Jakub**
Program: Informační technologie
Název: **Real-time řízení robota Melfa pomocí ROS**
Real Time Melfa Controlling Using ROS
Kategorie: Umělá inteligence

Zadání:

1. Seznamte se s robotickým pracovištěm vybaveným ramenem Melfa a programováním pomocí Robotického operačního systému (ROS) a simulačního prostředí Gazebo.
2. Navrhněte rozhraní mezi ROS a ramenem Melfa, které umožní řídit rameno v reálném čase. Pro plánování dráhy použijte knihovnu MoveIt. Vytvořte také funkční model pro simulování pracoviště v prostředí Gazebo.
3. Navržené rozhraní implementujte.
4. Knihovnu otestujte na sadě úkolů, např. pohyb z bodu A do bodu B, opakovaný pohyb ze základní polohy ramene do určeného bodu, pohyb z určeného bodu do řady bodů ležících v konstantních rozestupech na přímce. Body musí být voleny tak, aby se při pohybu ramene zapojily všechny motory.

Literatura:

- Howie Choset et al., Principles of Robot Motion, 2005, ISN 0-262-03327-5.
- Robotický operační systém, www.ros.org, [cit. 15.10.2020]

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 2020
Datum odevzdání: 12. května 2021
Datum schválení: 12. listopadu 2020

Abstrakt

Cieľom tejto bakalárskej práce je preskúmať možnosti ovládania priemyselného robotického ramena Mitsubishi Melfa RV-6SL pomocou ROS (Robotického operačného systému) a následná implementácia týchto poznatkov do funkčnej aplikácie. Táto práca popisuje čitateľovi nie len teoretické základy práce, ale aj spôsob vytvorenia konfiguračných balíčkov, potrebných pre simulovanie robotického ramena v simulačnom prostredí Gazebo, ako aj riadenie reálneho robotického ramena v reálnom čase. Text taktiež popisuje plánovanie trasy pomocou knižnice MoveIt, na základe polôh robotického ramena zvolených vo vizualizačnom prostredí Rviz.

Abstract

The goal of this bachelor thesis is to explore the possibilities of controlling the industrial robotic arm Mitsubishi Melfa RV-6SL using ROS (Robot Operating System) and the subsequent implementation of this knowledge into a functional application. This work describes to the reader not only the theoretical foundations of the work, but also how to create the configuration packages needed to simulate the robotic arm in the simulation environment Gazebo, as well as real-time control of the real robotic arm. The text also describes path planning using the MoveIt library, based on poses of robotic arm selected in the Rviz visualization environment.

Kľúčové slová

Mitsubishi, Melfa, RV-6SL, Gazebo, Rviz, realtime, plánovanie dráhy,robotické rameno, ROS, MoveIt, manipulátor

Keywords

Mitsubishi, Melfa, RV-6SL, ROS, MoveIt, Gazebo, Rviz, realtime, path planning, robotic arm, manipulator

Citácia

LIŠKA, Jakub. *Real-time řízení robota Melfa pomocí ROS*. Brno, 2021. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Jaroslav Rozman, Ph.D.

Real-time řízení robota Melfa pomocí ROS

Prehlásenie

Prehlasujem, že túto bakalársku prácu som vypracoval samostatne pod vedením pána Ing. Jaroslava Rozmana Ph.D. V tejto bakalárskej práci som uviedol všetky použité literárne pramene, publikácie a zdroje, z ktorých som čerpal.

.....
Jakub Liška
19. mája 2021

Poďakovanie

Rád by som poďakoval vedúcemu mojej bakalárskej práce pánovi Ing. Jaroslavovi Rozmanovi Ph.D za jeho trpezlivosť pri vypracovaní a testovaní implementácie tejto bakalárskej práce. Rád by som poďakoval aj rodičom a sestre za podporu. Taktiež by som sa touto cestou chcel poďakovať Bc. Antonovi Fircovi za zapožičanie výpočtovej techniky, potrebnej pre dokončenie a otestovanie výslednej implementácie.

Obsah

Kapitola 1

Úvod

Priemysel v súčasnej dobe zohráva veľmi dôležitú úlohu vo svetovej ekonomike. Aby bola zabezpečená vysoká efektivita a kvalita priemyselnej výroby, je kladený veľký dôraz na priemyselnú automatizáciu, ktorá pomáha tieto ciele úspešne napĺňať. Hlavným cieľom priemyselnej automatizácie je zavádzanie nových výrobných postupov, robotizácie a mechanizácie do procesu výroby. To by však nebolo možné, ak by v tejto oblasti neexistoval neustáli pokrok na poli vývoja týchto oblastí.

Môže sa zdať, že myšlienka automatizácie a predovšetkým robotizácie je výdobytkom nedávnej minulosti, no je tomu tak, že s touto myšlienkou sa začalo ľudstvo pohrávať už oveľa skôr. Jedným takým človekom bol napríklad aj český autor sci-fi románov Karel Čapek[52], ktorý vo svojom románe *R.U.R.*[47] po prvýkrát použil slovo *robot*[54]. Aj keď toto slovo vzniklo pred viac ako sto rokmi, dodnes ho používame pre pomenovanie rôznych strojov, ktorých úlohou je nahrádzať monotónnu činnosť vykonávanú človekom, vďaka čomu vzniklo celé odvetvie nazvané *robotizácia*.

V prípade robotizácie dnešnej doby sa jedná predovšetkým vývoj a integráciu robotických manipulátorov a ramien do výrobných procesov. Aktuálne v tomto odvetví pôsobí niekoľko desiatok veľkých, ale aj menších firiem (napríklad Fanuc, Kuka, ABB, Mitsubishi, a mnohé ďalšie), ako aj rôzne nezávislé open-source projekty, ktoré sa podieľajú na vývoji a raste tohoto dôležitého odvetvia.

Cieľom tejto práce je naplánovanie trasy a riadenie robotického ramena Mitsubishi Melfa RV-6SL za pomoci knižníc vytvorených týmito open-source projektami. Pre naplánovanie trasy robotického ramena je použitá knižnica *MoveIt!*[27], pre simuláciu robotického ramena simulačný software *Gazebo*[11] a o riadenie sa starajú knižnice vytvorené projektom *ROS (Robot Operating System)*[29]. Pre komunikáciu s reálnym ramenom je následne využitý voľne dostupný ovládač `melfa_driver`[44], ktorý je súčasťou balíčka `melfa_robot`[45].

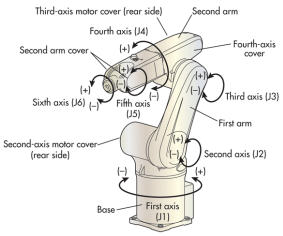
Výsledný text práce je možné rozdeliť na niekoľko častí. V prvej časti práce budú čitateľovi vysvetlené dôležité pojmy, predstavené robotické rameno, Robotický operačný systém (ROS)[29] a všetky potrebné nástroje. V ďalšej časti bude čitateľovi priblížený spôsob plánovania trasy robotického ramena a komunikácia s reálnym ramenom Mitsubishi Melfa RV-6SL. Posledná časť je venovaná predovšetkým implementačným detailom a testovaniu.

Kapitola 2

Robotické rameno

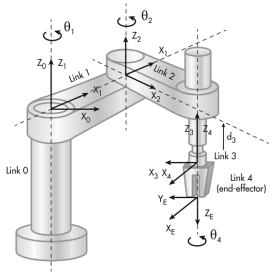
V priemyselnej robotike poznáme rôzne typy robotov[54], označovaných tiež robotické ramená alebo robotické manipulátory. Tieto je možné ďalej deliť na kategórie podľa rôznych kritérií. V súčasnej dobe sa pri priemyselných robotoch využívajú tieto tri hlavné parametre, pomocou ktorých ich charakterizujeme:

- **Typ konštrukcie:** Podľa typu aplikácie volíme vhodný typ konštrukcie tela robota. Tento parameter taktiež určuje rozsah pohybu[8][12].
  - **Angulárne roboty** - Najčastejšie využívaný typ. Jedná sa o typ robota so sériovou kinematikou 2. Telo robota sa skladá z niekoľkých článkov pospájaných osami do série.



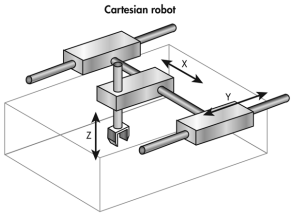
Obr. 2.1: Angulárny robot. Prevzaté z [12]

  - **SCARA roboty** - Typ robota so sériovou kinematikou 2, určený predovšetkým pre jednoduché aplikácie pre prácu s bremenom. Vyznačujú sa vysokou rýchlosťou pohybu.



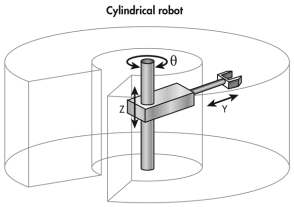
Obr. 2.2: SCARA robot. Prevzaté z [12]

– **Kartézske roboty** - Alebo tiež známe ako 3-osé manipulátory. Tento typ robotických manipulátorov so sériovou kinematikou 2 sa využíva pri špecifických aplikáciách.



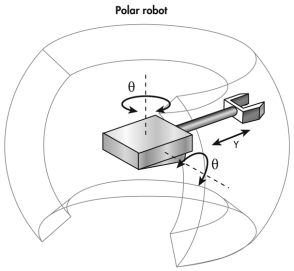
Obr. 2.3: Kartézsky robot. Prevzaté z [12]

– **Cylindrické roboty** - Málo využívaný typ. Jedná sa o typ robota so sériovou kinematikou 2. Charakteristický je typ pohybu v priestore, kde je jeden rotačný a dva lineárne pohyby.



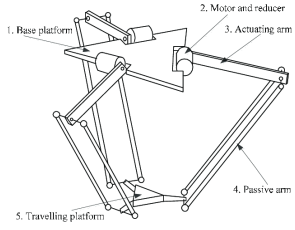
Obr. 2.4: Cylindrický robot. Prevzaté z [12]

– **Sférické roboty** - Málo využívaný typ. Jedná sa o typ robota so sériovou kinematikou 2. Charakteristický je typ pohybu v priestore, kde sú dva rotačné a jeden lineárny pohyb.



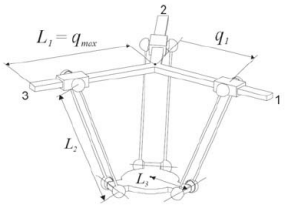
Obr. 2.5: Sférický robot. Prevzaté z [12]

– **Delta roboty** - Typ robota s paralelnou kinematikou 2. Vyznačuje sa vysokou rýchlosťou pohybu, najčastejšie sa využívajú pri práci s bremenom štýlom zdvihni a polož.



Obr. 2.6: Delta robot. Prevzaté z [20]

– **Tri/Hexa-pod** - Typ robota s paralelnou kinematikou 2. Obdoba typu delta. Konštrukčne sa líši od typu delta využitím lineárnych vedení pre pohyb koncového bodu.



Obr. 2.7: Tri-pod robot. Prevzaté z [21]

- **Nosnosť a dosah:** Konštrukčná nosnosť robotického zariadenia na koncovom bode zariadenia pre montáž nástroja a využiteľné pokrytie rozsahu pohybu s touto nosnosťou.
- **Úroveň kolaboratívnosti:** V súčasnej dobe vysoko populárne kritérium, ktoré určuje, do akej miery je priemyselný robot nebezpečný pre svoje okolie pri vykonávaní činnosti, pre človeka v jeho pracovnom priestore. Toto kritérium určujeme na základe štandardov **ISO 10218**[13][14] a **ISO/TS 15066**[15].

Kinematika robotických ramien a manipulátorov

V predchádzajúcej časti textu 2 sme si predstavili, aké typy konštrukcie robotických ramien a manipulátorov existujú. Pri popise sme využívali pojmu *kinematika*. Táto časť textu je prevzatá z [56].

Pomocou kinematiky popisujeme vzťahy medzi polohou, rýchlosťou a zrýchlením skupiny pevných telies, v našom prípade robotických ramien a manipulátorov. Kinematika zohráva dôležitú úlohu pri plánovaní trajektórie robotických ramien a manipulátorov, nakoľko nám popisuje spôsob a možnosti pohybu zariadenia v priestore. Pomocou kinematiky vieme popísať, ako sa prejaví zmena natočenia kĺbov robotického manipulátoru na výslednej pozícii koncového bodu (nástroja) v karteziánskom priestore. Riadiaci systém robotických ramien a manipulátorov však pre zmenu pozície vyžaduje od plánovača hodnoty natočenie jednotlivých kĺbov, ktorých kombináciou docielime výslednú pozíciu.

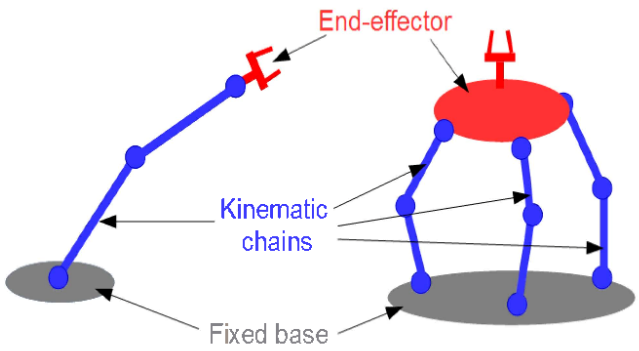
Pomocou kinematiky je tak možné vykonávať prepočet medzi týmito reprezentáciami podľa potreby priestorovej reprezentácie. Rozlišujeme tak dve transformácie:

- Priama kinematika (Forward kinematics) - využíva sa priama transformácia, kedy na základe hodnôt natočenia kĺbov robotického manipulátoru, napríklad $\theta_1, \theta_2, \theta_3, \theta_4$, θ_5, θ_6, vieme určiť súradnice bodu v karteziánskom priestore *(x, y, z)*.
- Inverzná kinematika (Inverse kinematics) - využíva sa inverzná transformácia, kedy zo známych karteziánskych súradníc bodu v priestore *(x, y, z)* určíme zodpovedajúce natočenia kĺbov robotického manipulátoru, napríklad $\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6$.

Inverzná kinematika na rozdiel od priamej kinematiky nezaručuje jednoznačné výsledky. Pri určitých bodoch v karteziánskom priestore je možné tieto body, s použitím inverznej kinematiky, dosiahnúť niekoľkými kombináciami natočenia kĺbov. Naopak tiež môže nastať situácia, kedy neexistuje taká kombinácia natočenia kĺbov, aby sme tento bod dosiahli. Všetko záleží od typu konštrukcie robota, počtu kĺbov, rozsahu pohybu a dosahu manipulátoru, ako aj rôzne externé obmedzenia formou prekážok v priestore.

V predchádzajúcej časti textu 2 sme taktiež v prípade kinematiky manipulátorov rozlišovali sériovú a paralelnú kinematiku.

- Sériová kinematika - typ kinematiky, ako už názov napovedá, kedy sú segmenty pospájané pomocou kĺbov do série. To znamená, že jeden kĺb spája dva segmenty a jeden segment je pripojený k dvom kĺbom, ak sa nejedná o koncové segmenty (základňu, segment určený pre montáž nástroja), ku ktorým v bežných situáciách prislúcha len jeden kĺb.
- Paralelná kinematika - typ kinematiky, kedy sú všetky poháňané kĺby súčasťou jedného segmentu, väčšinou základne a koncový segment je spojený s týmito kĺbmi pomocou niekoľkých sérii pasívnych segmentov a kĺbov, ktoré sú voči sebe paralelné.



Obr. 2.8: Sériová (vľavo) a paralelná (vpravo) kinematika manipulátora. Prevzaté z [57]

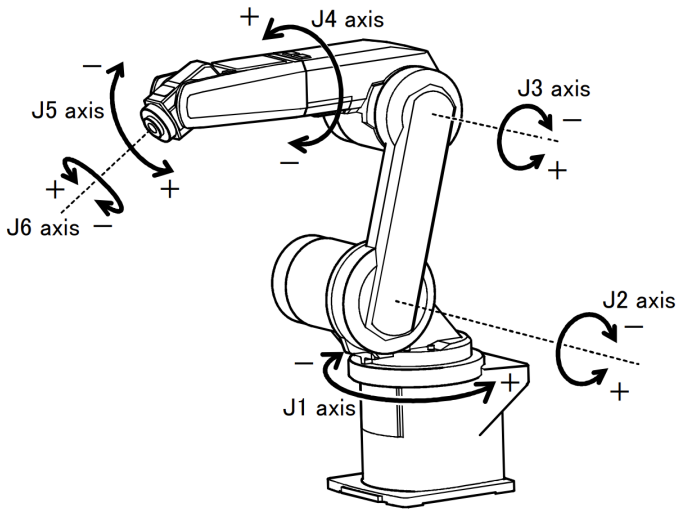
2.1 Robotické rameno Mitsubishi Melfa RV-6SL

Robotické pracovisko na Fakulte informačných technológii VUT v Brne je vybavené robotickým ramenom Mitsubishi Melfa RV-6SL, ktoré je vybavené nástrojom, elektronickým gripprom (uchopovačom) Schunk PG-70[43]. Tento gripper nie je súčasťou implementácie ovládania ramena, preto mu bude v ďalších častiach textu venovaná len minimálna pozornosť, avšak nie je pre prácu zanedbateľný, nakoľko pri ovládaní robotického ramena je nutné jeho prítomnosť brať do úvahy. Ak by sme tak neurobili, mohlo by dôjsť k poškodeniu grippera alebo samotného robotického ramena, čo nie je žiadúce.

2.1.1 Popis robotického ramena

Robotické rameno Mitsubishi Melfa RV-6SL je angulárneho typu so šiestimi osami (kĺbmi) a siedmimi segmentmi. Jeden zo segmentov slúži ako základňa a je pripevnená do podložky. Zvyšné segmenty sú pohyblivé. Ich pohyb zabezpečujú servomotory s inkrementálnymi enkodérmi, ktoré sú spriahnuté s prevodovkami.

Každý z kĺbov má špecifické vlastnosti, najmä rozsah pohybu v kladnom a zápornom smere od nulovej polohy, no nesmieme zabudnúť aj na rýchlosť otáčania a maximálnu možnú silu, ktorú dokáže vyvinúť. Tieto, ale aj ďalšie informácie, ako sú rozmery, graf dosahu a nosnosti robota môžeme nájsť v manuáli, ktorý výrobca poskytuje v elektronickej alebo papierovej podobe.[3]



Obr. 2.9: Mitsubishi Melfa RV-6SL. Prevzaté z [4]

V prípade robotického ramena Mitsubishi Melfa RV-6SL udáva výrobca maximálnu nosnosť 6kg vrátane nástroja a nie je určený pre prácu v otvorenom priestore s človekom pri nasadení vo výrobnom procese.

Preto pri práci s týmto robotickým ramenom v otvorených laboratórnych podmienkach je nutné dodržiavať zásady bezpečnosti a riadne vopred zvážiť riziká, ktoré môžu nastať pri spustení pohybu robotického ramena.

Kontrolér robotického ramena

Kontrolér robotického ramena je nevyhnutná súčasť celého zariadenia. Bez tejto časti by samostatné rameno bolo len kus nehybnej masy kovu a elektroniky.

Pod pojmom kontrolér tak označujeme časť zariadenia, ktorá v sebe zahŕňa riadiacu a výkonnostnú elektroniku. Táto elektronika nie je umiestnená priamo v tele robotického ramena, vzhľadom na svoje rozmery, ale je umiestnená v samostatnom boxe alebo skrini (rôznych rozmerov). Pre spojenie ramena a kontroléra sa využívajú špeciálne niekoľko žilové káble, ktoré sú schopné prenášať silovú a logickú časť riadenia robotického ramena.

Podstatné komponenty sa nachádzajú samozrejme vo vnútri zariadenia. Pojmom riadiaca elektronika sme označili v našom prípade matičnú dosku s procesorom, pamäťami a ďalšími potrebnými komponentami, ktoré sú nutné k behu systému a riadiacich programov robotického ramena. Výkonnostnou cestou následne označujeme časť kontroléru, ktorá obsahuje komponenty potrebné k napájaniu servomotorov a ostatných súčasti zariadenia.

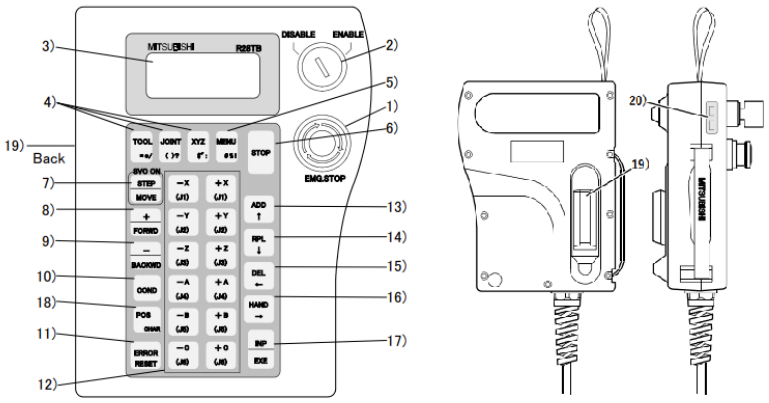
Obr. 2.10: Mitsubishi Melfa CR2B-574. Prevzaté z [3]

V našom prípade je robotické rameno Mitsubishi Melfa RV-6lS vybavené kontrolórom Mitsubishi Melfa CR2B-574, ktorý je menším z dvojice kompatibilných kontrolérov [3]. Na čelnej strane tohto kontroléru sa nachádza jednoduchý ovládací panel so stavovým displejom a niekoľkými tlačidlami pre ovládanie základných funkcií, núdzové tlačidlo, prepínač pracovných režimov ovládaný servisným kľúčom a konektor pre pripojenie Teaching pendantu 2.1.1. Na zadnej strane sa následne nachádzajú potrebné konektory pre pripojenie k samotnému ramenu a iným externým zariadeniam.

Teaching pendant

Teaching pendant, v našom prípade model R28TB, je periférne zariadenie, ktoré je štandardne dodávané spolu s robotickým ramenom a kontrolérom. Toto zariadenie je prenosné a pripája sa ku kontroléru cez konektor v prednej časti. Pre bežnú prevádzku nie je potrebné a v prípade kontroléru Melfa CR2B-574 je možné ho odpojiť.

Účelom teaching pendantu je rozšírenie kontroléru o pokročilý ovládací panel obsahujúci klávesnicu a niekoľkoriadkový monochromatický displej. Využíva sa najmä pri vytváraní a editovaní programov, ktoré sa v robotike nazýva teaching (učenie), servisných úkonoch a diagnostike zariadenia.



Obr. 2.11: Mitsubishi Melfa R28TB. Prevzaté z [2]

2.1.2 Programovací jazyk Melfa-Basic

Pre programovanie robotických ramien a manipulátorov z rodiny Mitsubishi Melfa je využívaný proprietárny programovací jazyk *Melfa-Basic*. Pre potreby jednoduchého vytvárania riadiaceho programu za pomoci Teaching pendantu 2.1.1 sa jedná o jazyk symbolických inštrukcií, rovnako ako v prípade programovacieho jazyka *BASIC* [50], ktorému je do značnej miery podobný svojou syntaxou.

Jednotlivé inštrukcie sa vkladajú na samostatné riadky a následne sú v tomto poradí vykonávané. Každý riadok programu musí obsahovať číslo riadku, nasledované inštrukciou a jej prípadnými argumentami, napríklad:

```
10 MOV P1 TYPE 1,0
20 MOV J1
30 MOV (PLT 1,10),100.0 WTH M_OUT(17)=1
40 MOV P4+P5,50.0 TYPE 0,0 WTHIF M_IN(18)=1,M_OUT(20)=1
```

Výpis 2.1: Referenčný príklad inštrukcie MOV. Prevzaté z [5]

V prípade robotického ramena Melfa RV-6SL sa využíva štvrtá verzia tohoto programovacieho jazyka *Melfa-Basic IV*.

2.1.3 Komunikácia s externým zariadením

Cieľom tejto bakalárskej práce nie je riadenie robotického ramena za pomoci trasy vytvorenej v programovacom jazyku *Melfa-Basic IV*, ale riadenie za pomoci externého zariadenia. Externým zariadením je v našom prípade osobný počítač vybavený potrebným softwarom, ktorý je k robotickému ramenu, respektíve jeho kontroléru pripojený pomocou sieťového rozhrania Ethernet a komunikuje pomocou protokolu UDP 5.3.

Pre komunikáciu s externým zariadením je nutné vytvoriť jednoduchý program v Teaching pendante robotického ramena, ktorý zabezpečí po spustení otvorenie sieťového rozhrania pre komunikáciu a následne spracovanie riadiacich príkazov v reálnom čase od externého zariadenia.

Potrebný program je nasledovný:

```
10 OVRD 10
20 OPEN "ENET:192.168.0.2" AS #1
30 MXT 1,1
40 END
```

Výpis 2.2: Real-time program v robotickom ramene. Prevzaté z [45]

Aj keď sa jedná o jednoduchý program, zabezpečuje všetko potrebné pre komunikáciu a ovládanie ramena za pomoci externého zariadenia. Funkcionalita programu je nasledovná:

- Na riadku 10 je nastavený Override (softwarové obmedzenie) maximálnej rýchlosti na 10 percent. [1]
- Na riadku 20 dochádza k otvoreniu sieťového rozhrania Ethernet a nastaveniu IP adresy zariadenia, s ktorým bude následne kontrolér ramena komunikovať. Komunikácia bude následne v kontroléri reprezentovaná ako súbor s označením 1.
- Na riadku 30 sa nachádza inštrukcia MXT *(Move External)*. Táto inštrukcia zabezpečuje real-time komunikáciu s externým zariadením, vykonávanie inštrukcií zaslaných kontroléru a podávanie spätnej väzby o stave robotického ramena. V našom prípade je inštrukcia v tvare MXT 1,1. Prvý argument udáva, pomocou ktorého súboru sa interne predávajú dáta (v našom prípade 1). Druhý argument udáva, aký typ pozičných dát sa zasiela v komunikácií. Hodnota 1 odpovedá typu dát, ktoré reprezentujú natočenie jednotlivých kĺbov. V manuáli [5] sú popísane aj ďalšie možnosti.
- Na riadku 40 sa nachádza inštrukcia END, označujúca koniec programu.

2.1.4 3D model

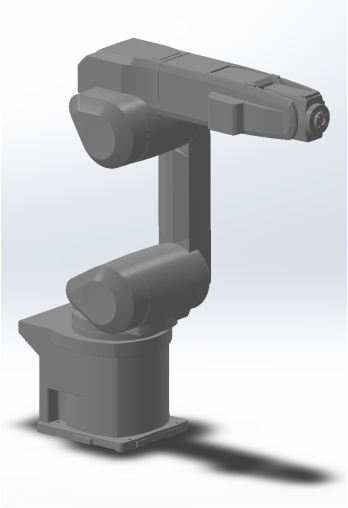
Cieľom tejto bakalárskej práce je okrem samotného riadenia reálneho robotického ramena aj simulácia tohto ramena v simulačnom softwari Gazebo3.4 a RVIZ 3.3. Tieto simulačné prostredia však nedisponujú žiadnou knižnicou modelov a model je potrebné zabezpečiť samostatne.

Pre tieto účely je možné si model vytvoriť svojpomocne, ak sa jedná napríklad o doma vyrobené robotické rameno, no v našom prípade využívame priemyselné robotické rameno a preto táto náročná činnosť odpadá. Renomovaný výrobcovia bežne ponúkajú zadarmo detailné modely svojich robotických ramien na svojich webových stránkach, pripadne na vyžiadanie cez obchodné oddelenia. Model nášho robotického ramena je možné stiahnuť z webového úložiska výrobcu po zaregistrovaní sa na jeho webových stránkach [24].

V ponuke je niekoľko typov formátov, no pre naše potreby je vhodné zvoliť model vo formáte STEP, nakoľko je tento formát možné importovať do CAD softwaru Solidworks 2019 [9]. Voľba CAD softwaru bude bližšie objasnená v časti textu, ktorá sa zaoberá vytvorením balíčka potrebného pre prácu s nástrojmi frameworku ROS 2.1.5. Po importe modelu do vizualizačného softwaru sa nám zobrazí 3D model robotického ramena Melfa RV-6SL.

Ďalším krokom je úprava 3D modelu pre účely simulácie. Formát STEP [51] nepodporuje pohyblivé modely a taktiež nenesie v sebe ani informáciu o hmotnosti a materiále telesa, ak do modelu explicitne konštruktér nevloží tieto informácie počas exportu.

[1] Neskôr pri testovaní a následnom hlbšom skúmaní bolo z manuálu [5] zistené, že toto obmedzenie sa nevzťahuje na inštrukciu MXT.



Obr. 2.12: 3D model Mitsubishi Melfa RV-6SL zobrazený v Solidworks

V našom prípade tak konštruktér neučinil, nakoľko sa pri týchto modeloch nepredpokladá využitie v simulátoroch. Model je preto vytvorený len ako objemové teleso. Informáciu o hmotnosti a pohyblivé kĺby je preto potrebné doplniť. Rovnako tak do modelu doplníme nástroj, gripper Schunk PG-70. Model tohto grippera je možné stiahnuť opäť na webe výrobcu, obdobne aj pri robotickom ramene.

Pohyblivé kĺby a gripper

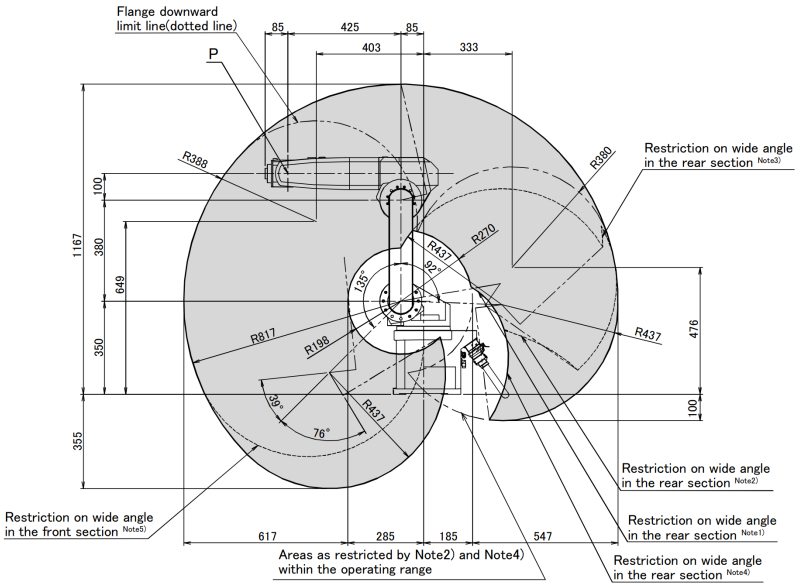
Bežné univerzálne formáty využívané pri CAD modeloch, určené pre prenositeľnosť modelov medzi CAD softwarmi rôznych výrobcov nepodporujú veľké množstvo informácií, napríklad pohyblivé kĺby. Toto je možné napraviť úpravou modelu a následným uložením v proprietárnom formáte. V prípade Solidworks sa jedná o formát SLDASM.

Vytvorenie pohyblivých kĺbov je možné niekoľkými spôsobmi. V našom prípade sme zvolili spôsob vyexportovania podzostáv segmentov z pôvodného modelu vo formáte STEP a následnému spätnému poskladaniu v novom projekte, kde sme zadefinovali potrebné väzby medzi segmentami. Pri tomto spôsobe je potrebné venovať veľkú pozornosť správnemu zarovnaniu dielov a obzvlášť veľkú pozornosť je potrebné venovať osiam rotácie kĺbov. Pri väčšine kĺbov s týmto nebol obzvlášť veľký problém, nakoľko model obsahoval dostatočné množstvo referenčných bodov. Problém nastal pri 4. kĺbe (osi) robota. V tomto prípade je os mierne posunutá voči rezu segmentu, ktorý má navyše značne nepravidelný tvar. Tento problém sa podarilo vyriešiť zadefinovaním pomocného bodu.

Pri tomto procese skladania modelu je vhodné vložiť do modelu tiež nástroj. V našom prípade sa nástroj vkladal dodatočne, nakoľko pri počiatočnom procese sme nemali bližšie informácie o rozmeroch uchopovacích prstov a montážnej platne. Po získaní týchto informácií sme následne namodelovali montážnu platňu grippera a prsty. V prípade prstov sme zvolili postup namodelovania a nahradenia pomocou boundry boxov, z dôvodu ich zložitého tvaru. Pre účely simulácie je táto informácia dostatočná, presný tvar by mal minimálny prínos. Účelom bolo do modelu zaniesť informáciu, ktorá nám umožní sa vyhnúť poškodeniu hardwaru v reálnych podmienkach. Taktiež sme do modelu pridali jednoduchý valcový objekt, ktorý reprezentuje káblovú prechodku a kábel vedúci do grippru, aby nemohlo dôjsť k jeho poškodeniu v prípade pohybu v tesnej blízkosti iných segmentov robotického ramena.

Pri opätovnom skladaní modelu je vhodné tiež preveriť správnu orientáciu a nulové polohy kĺbov. Nulové polohy kĺbov je nutné dodržať, čím sa vyhneme problémom v ďalších častiach práce.

V našom prípade je robotické rameno v originálnom STEP modeli 2.12, ale aj v oficiálnych materiáloch a dokumentácii vyobrazené v polohe, kedy sú polohy kĺbov zarovnané na kalibračných ryskách. Kalibračné rysky však nemusia vždy odpovedať nulovým polohám kĺbov. V prípade nášho robotického ramena Mitsubishi Melfa RV-6SL tento prípad nastal. Ak sú segmenty spojené tretím kĺbom zarovnané na kalibračnej ryske, poloha kĺbu je vychýlená od nulového bodu o 90°v kladnom smere rotácie[2]. Táto informácia nie je v texte štandardného manuálu spomenutá, no je možné ju odvodiť z obrázka rozsahu a dosahu pohybu 2.13 v kombinácii s tabuľkou maximálnych rozsahov pohybu kĺbov v manuáli[3].



Obr. 2.13: Mitsubishi Melfa RV-6SL, rozsah pohybu. Prevzaté z [3]

Hmotnosti segmentov

Pre korektné riadenie a plánovanie trasy robotického ramena je nutné brať do úvahy aj hmotnosti samotných segmentov. Bližší význam je popísaný v nasledujúcich častiach textu [38] a 4.

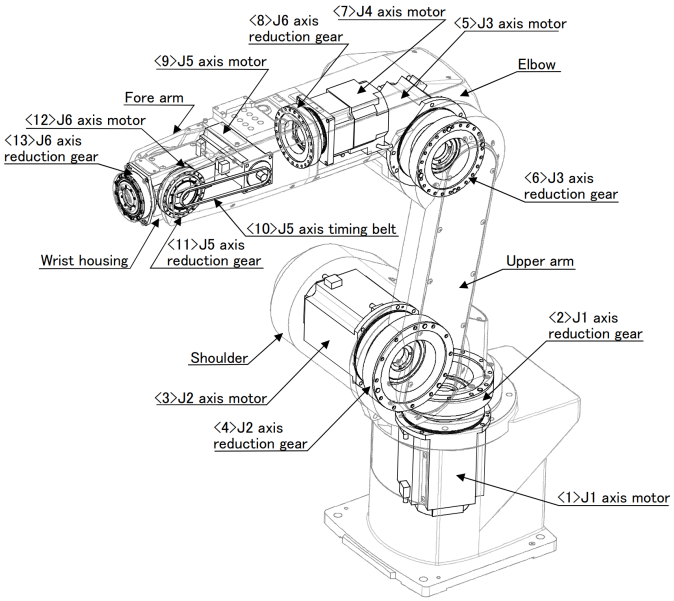
V prípade nášho modelu nie sú hmotnosti jednotlivých segmentov definované, preto ich je potrebné doplniť. Ak by sa jednalo o doma vyrobené robotické rameno, hmotnosti jednotlivých segmentov by sme do modelu vedeli zaniesť už počas jeho vytvárania pomocou definovania materiálov použitých na výrobu alebo prevážením samotných komponentov. Naše robotické rameno je však priemyselného typu a jeho rozobratie neprichádza do úvahy vďaka zložitosti konštrukcie. Rozobratie by tiež znamenalo nutnosť opätovnej kalibrácie celého ramena odborne školeným technikom, aby bola zaručená správna funkčnosť a presnosť ramena.

[2]Toto sme neskôr overili počas testovania, kedy po zarovnaní rysiek na treťom kĺbe/osi robotické rameno zobrazovalo polohu +90°na displeji Teaching pendantu

Hmotnosti sme preto do výslednej implementácie odhadli. Naše robotické rameno má celkovú hmotnosť približne 60kg podľa výrobného štítku na tele ramena. Presné hmotnosť ramena je závislá na príplatkovej výbave. Naše robotické rameno však žiadnou takouto výbavou nedisponuje, preto budeme brať 60kg ako konečnú hmotnosť, samozrejme bez nástroja.

Samotný CAD software Solidworks v nástroji pre výpočet hmotnosti vypočítal hmotnosť ramena na približne 32.3kg, nakoľko vychádzal z celkového objemu telesa a ako defaultnú konštantu hustoty telesa použil hustotu 1000kg/m^3. Po zadefinovaní inej hustoty alebo konkrétneho typu materiálu pre jednotlivé komponenty následne nastroj prepočítava hmotnosť na základe týchto parametrov.

V našom prípade je však nutné brať do úvahy, že jednotlivé segmenty môžu mať hmotnosť ovplyvnenú nie len samotným materiálom, ale aj prítomnosťou dutín, kabeláže a servomotorov s prevodovkami 2.14, ktoré sa podieľajú na značnej časti hmotnosti celého robotického ramena, nakoľko telo ramena je vyrobené z neznámej ľahkej zliatiny.



Obr. 2.14: Mitsubishi Melfa RV-6SL, vnútorné usporiadanie servomotorov. Prevzaté z [4]

Na základe servisného manuálu sme následne vyhľadali v servisnom portáli [23] väčšinu vnútorných komponentov. Servisný portál s náhradnými dielmi v prípade firmy Mitsubishi obsahuje nie len relatívne detailné fotografie, ale aj špecifikáciu dielov, medzi ktorými udáva aj hmotnosť konkrétneho dielu. Na portáli sa však nenachádzali všetky diely, z ktorých je reálne rameno poskladané, ale len prevažná časť vnútorných komponentov, ktoré sa môžu časom opotrebovať a poškodiť.

Na základe týchto informácií je možné vypočítať rozloženie hmotnosti vnútorných komponentov jednotlivých segmentov robotického ramena, tak ako na obrázku 2.14, ktoré spolu vážia približne 40.8kg, čo tvorí približne dve tretiny hmotnosti celého robotického ramena. Zvyšných 19.2kg bolo následne pomerovo rozdelených medzi jednotlivé segmenty na základe objemu segmentov získaných zo Solidworks s miernymi korekciami na základe prihliadnutia na fakt, že niektoré časti sú tvorené plastovými dielmi alebo obsahujú veľké dutiny.

Po určení hmotnosti boli následne vypočítané priemerné hustoty pre jednotlivé segmenty, pomocou vzorca na výpočet hustoty (2.1) a objemov získaných zo Solidworks. Získané hustoty boli následne využité pri definícii nových materiálov, ktoré boli následne priradené jednotlivým segmentom.

$$\rho = m/V \tag{2.1}$$

Rovnakým spôsobom boli získane aj hodnoty pre zadefinovanie hmotnosti grippera. Na stránke výrobcu [43] je udaná hmotnosť 1.4kg, hmotnosť montážnej platne bola vypočítaná pomocou Solidworks, pomocou knižnice materiálov. Platňa je vyrobená z hliníka. Hmotnosť uchopovacích prstov bola odhadnutá. V nasledujúcej tabuľke 2.1 sú uvedené výsledné hmotnosti jednotlivých segmentov a kompletného grippera.

Zostava	Hmotnosť (g)
Základňa	16300
Segment 1	12570
Segment 2	7580
Segment 3	13800
Segment 4	7050
Segment 5	2580
Segment 6	60
Gripper	1918
Spolu	61 858

Tabuľka 2.1: Tabuľka hmotností segmentov (zaokrúhlené na celé gramy)

Výsledný model

Po aplikovaní všetkých úprav spomínaných v predchádzajúcich podkapitolách, je potrebné výsledný model uložiť vo formáte SLDASM, ktorý dokáže uchovať všetky vykonané zmeny. Výsledný model po zarovnaní kĺbov do nulových polôh a osadení grippera je možné vidieť na obrázku 2.15.

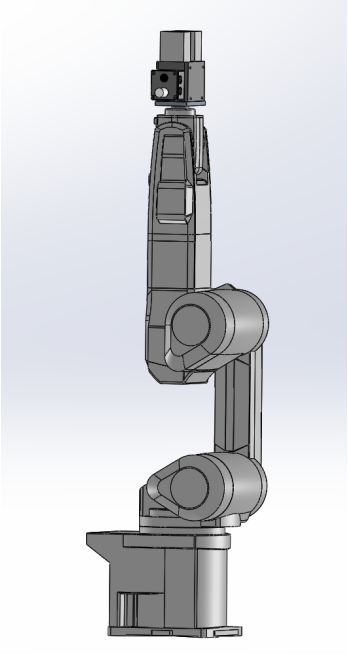
2.1.5 URDF

V predchádzajúcich častiach práce sme sa zaoberali vytvorením čo najpresnejšieho 3D modelu nášho robotického ramena a pracovali sme s týmto modelom v CAD softwary Solidworks. Výsledkom našej práce je pohyblivý model, ktorý je uložený v proprietárnom formáte využívanom v Solidworks.

Pre potreby využitia v simulačnom prostredí RVIZ, Gazebo a samotnom frameworku ROS, ktorými sa budú zaoberať ďalšie časti práce, nie je možné náš model v tomto formáte využiť. Preto je potrebné náš model exportovať do formátu, ktorý je kompatibilný s frameworkom ROS a jeho nástrojmi.

Formát URDF vo všeobecnosti

The Unified Robot Description Format (URDF) [38], voľne tiež preložiteľné ako univerzálny robota popisujúci formát, je textová špecifikácia robota, zapísaná pomocou značkovacieho jazyka XML [46]. Nemusí sa jednať nutne o robotické rameno. Obsahom takéhoto súboru



Obr. 2.15: 3D model Mitsubishi Melfa RV-6SL osadený gripperom

je detailný popis parametrov jednotlivých komponentov robota pomocou súboru elementov definovaných v štandarde URDF.

V textovej podobe sú zapísané parametre ako napríklad názov komponentu, jeho hmotnosť, orientácia, príslušnosť v hierarchii komponentov celého modelu, ale sú tu tiež dôležité informácie ako kinematika. Tieto informácie sú ďalej doplnené o 3D modely jednotlivých komponentov v univerzálnom CAD formáte STL [55]. Formát STL (Standard Tessellation Language) popisuje komponenty na základe plochy telesa. Tieto 3D dáta následne využívajú plánovacie knižnice pre plánovanie trasy, ale aj vizualizačné a simulačné nástroje pre potreby zobrazenia modelu robota a riešenie kolízií.

SolidWorks to URDF Exporter

V časti textu 2.1.4 venovanej 3d modelu ramena sme pre prácu s týmto modelom volili CAD software Solidworks. Dôvod tejto voľby sme si vtedy bližšie nešpecifikovali a pre úpravu modelu sme mohli zvoliť iný konkurenčný CAD software. Hlavným dôvodom tejto voľby je existencia plugin modulu s názvom SolidWorks to URDF Exporter, vytvoreného komunitou frameworku ROS. Navod na jeho pouzitie sa nachadza [31].

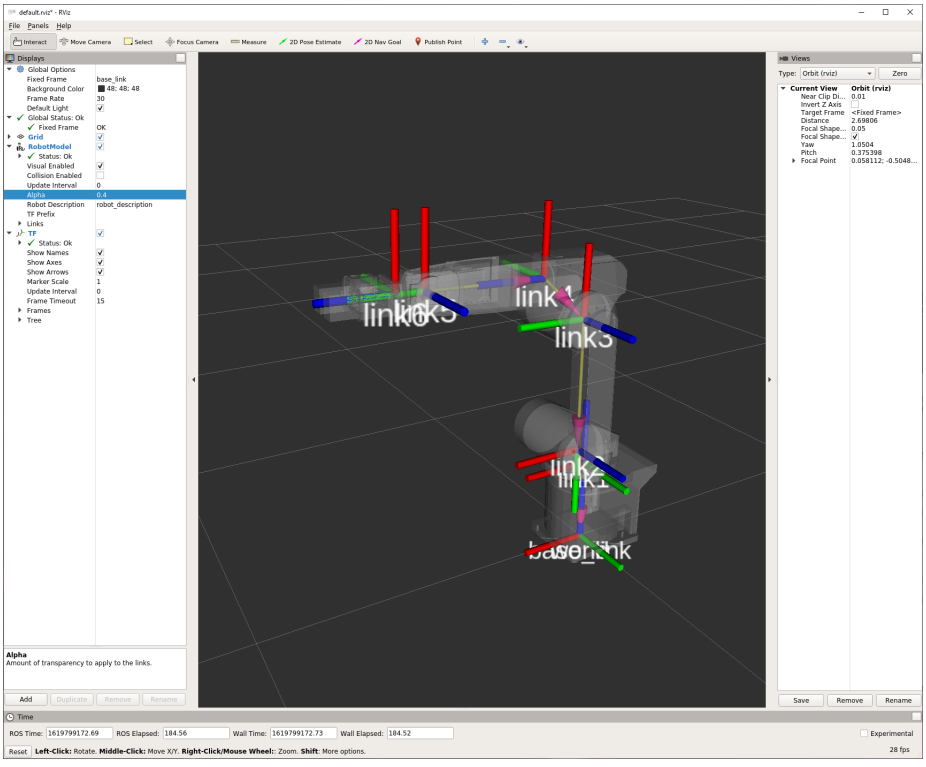
Tento modul alebo tiež nástroj po nainštalovaní do prostredia Solidworks umožňuje, ako už samotný názov napovedá, export 3D modelu do formátu URDF. Po spustení tohoto nástroja tak v niekoľkých krokoch vyplníme v tomto nástroji potrebné parametre, ktoré náš model neobsahuje. Na základe nami doplnených parametrov a parametrov získaných priamo z 3D modelu pomocou nástrojov obsiahnutých v Solidworks následne tento nastroj vygeneruje balíček, obsahujúci kompletný popis robota vo formáte URDF.

Denavit–Hartenberg parametre

Denavit–Hartenberg parametre [48] sú štyri parametre s určenou konvenciou, ktoré sa využívajú v robotike a slúžia na priraďovanie referenčných priestorov (framov) segmentom robotických manipulátorov so sériovou kinematikou. Tieto konvencie vznikli z dôvodu vytvárania transformácií súradnicového systému naprieč jednotlivými segmentami robotických manipulátorov.Jednotlivé súradnicové systémy majú svoje počiatky v kĺboch medzi jednotlivými segmentami. Orientácia osí sa určuje na základe predchádzajúceho súradnicového systému a osi otáčania kĺbu pomocou niekoľkých pravidiel.

V našom prípade toto využijeme pre úpravu orientácie týchto súradnicových systémov, nakoľko SolidWorks to URDF Exporter neurčil tieto orientácie správne a nekorešpondujú so systémom, ktorý je uvádzaný v manuláli [4].

Počiatočný bod pre počiatočný systém súradníc leží na priesečníku osi otáčania prvého kĺbu a spodnej časti základne, rovnako ako pri systéme, ktorý využíva rameno pri práci s originálnym kontrolérom. Koncový bod, počiatok súradnicového systému leží na osi otáčania šiesteho kĺbu v dutine pod prírubou grippera.



Obr. 2.16: Výsledné orientácie súradnicových systémov zobrazených v RViz.

2.1.6 Melfa RV-6SL – URDF balíček

Po splnení všetkých potrebných krokov sme vygenerovali pomocou nástroja SolidWorks to URDF Exporter výsledný balíček, obsahujúci popis nášho robotického ramena. Výsledný balíček s názvom MELFA_RV-6SL má nasledujúcu štruktúru:

```
melfa_rv-6sl
├── config
│   └── joint_names_melfa_rv-6sl.yaml
├── launch
│   ├── display.launch
│   ├── gazebo.launch
│   └── melfa_rv-6sl.rviz
├── meshes
│   ├── base_link.STL
│   ├── link1.STL
│   ├── link2.STL
│   ├── link3.STL
│   ├── link4.STL
│   ├── link5.STL
│   └── link6.STL
├── textures
├── urdf
│   ├── melfa_rv-6sl.csv
│   └── melfa_rv-6sl.urdf
├── CMakeLists.txt
├── export.log
└── package.xml
```

Výsledný balíček obsahuje okrem súboru vo formáte URDF aj jeho obdobu vo formáte CSV a ďalšie automaticky vygenerované súbory. Okrem spomínaných súborov obsahujúcich 3D modely komponentov, v našom prípade segmenty ramena, sú súčasťou balíčka ešte predkonfigurované spúšťacie súbory, pomocou ktorých je možné spustiť vizualizačné prostredie RVIZ a simulátor Gazebo. Taktiež sú súčasťou konfiguračné súbory nutné pre simuláciu alebo preklad samotného balíčka.

2.1.7 Štruktúra URDF súboru

Model robota aj reálny robot sú tvorené segmentami a kĺbmi. Popis modelu robota [33] je následne zapísaný v URDF súbore pomocou XML [46] značkovacieho jazyka a základná štruktúra je tvorená koreňovým elementom `<robot>`, ktorý obsahuje potomkov `<link>` (segment) a `<joint>` (kĺb).

```
<?xml version="1.0" encoding="utf-8" ?>
<robot name="nazov_robota">
    <link> ... </link>
    <link> ... </link>
    <link> ... </link>

    <joint> .... </joint>
```

```
    <joint> .... </joint>
    <joint> .... </joint>
</robot>
```

Výpis 2.3: Základná štruktúra URDF. Prevzaté z [33]

Element `<link>`

Tento element v URDF popisuje segment robotického ramena. Tento element obsahuje ďalšie vnorené elementy a ich atribúty. Konkrétne obsahuje tri elementy, ktoré popisujú zotrvačnosť, vlastnosti pre vizualizáciu a detekciu kolízií.

```
<link name="base_link">
    <inertial>
        <origin xyz="0.0013701 -6.4154E-05 0.12542" rpy="0 0 0" />
        <mass value="16.301" />
        <inertia ixx="0.16272" ixy="-1.3928E-05" ixz="0.00036162"
                 iyy="0.15649" iyz="-0.00011637" izz="0.11345" />
    </inertial>

    <visual>
        <origin xyz="0 0 0" rpy="0 0 0" />
        <geometry>
            <mesh filename="package://melfa_rv-6sl/meshes/base_link.STL" />
        </geometry>
        <material name="">
            <color rgba="0.75294 0.75294 0.75294 1" />
        </material>
    </visual>

    <collision>
        <origin xyz="0 0 0" rpy="0 0 0" />
        <geometry>
            <mesh filename="package://melfa_rv-6sl/meshes/base_link.STL" />
        </geometry>
    </collision>
</link>
```

Výpis 2.4: Príklad štruktúry elementu `<link>`.

Význam všetkých elementov a atribútov, ktoré môže obsahovať element `<link>` je nasledovný:

- `name` - atribút, názov segmentu
- `<inertial>` - vlastnosti zotrvačnosti telesa
  - `<origin>` - posun a rotácia ťažiska segmentu vzhľadom k jeho počiatku
    - `xyz` - hodnoty posunu v 3D podľa osí x,y,z [mm]
    - `rpy` - hodnoty rotácie v 3D podľa osí x,y,z [rad]
  - `<mass>` - hmotnosť telesa (segmentu)

`value` - hodnota hmotnosti [kg]

`<inertia>` - zotrvačnosť telesa

`ixx ixy ixz iyy iyz izz` - hodnoty zotrvačnosti, viď spôsob výpočtu 2.1.8

- `<visual>` - vlastnosti segmentu pre vizualizáciu robota

  `<origin>` - posun a rotácia segmentu vzhľadom k jeho počiatku

  `xyz` - hodnoty posunu v 3D podľa osí x,y,z [mm]

  `rpy` - hodnoty rotácie v 3D podľa osí x,y,z [rad]

  `<geometry>` - tvar segmentu, môže obsahovať jeden z nasledujúcich elementov

  `<box>` - tvar segmentu je kváder, atribúty sú dĺžky hrán [mm]

  `<cylinder>` - tvar segmentu je valec, atribúty sú dĺžka a polomer [mm]

  `<sphere>` - tvar segmentu je guľa, atribút je polomer [mm]

  `<mesh>` - tvar segmentu je uložený v STL súbore, atribút je cesta k umiestneniu

  `<material>` - farba alebo textúra povrchu, `name` - atribút, pomenovanie materiálu

  `<texture>` - textúra materiálu, atribút je cesta k súboru

  `<color>` - farba, atribút `rgba` pre nastavenie farby

- `<collision>` - vlastnosti geometrie segmentu pre potreby plánovania a detekcie kolízií

  `<origin>` - posun a rotácia segmentu vzhľadom k jeho počiatku

  `xyz` - hodnoty posunu v 3D podľa osí x,y,z [mm]

  `rpy` - hodnoty rotácie v 3D podľa osí x,y,z [rad]

  `<geometry>` - tvar segmentu, môže obsahovať jeden z nasledujúcich elementov

  `<box>` - tvar segmentu je kváder, atribúty sú dĺžky hrán [mm]

  `<cylinder>` - tvar segmentu je valec, atribúty sú dĺžka a polomer [mm]

  `<sphere>` - tvar segmentu je guľa, atribút je polomer [mm]

  `<mesh>` - tvar segmentu je uložený v STL súbore, atribút je cesta k umiestneniu

Element `<joint>`

Tento element v URDF popisuje kĺb robotického ramena, ktorý sa nachádza medzi dvoma segmentami. Tento element obsahuje ďalšie vnorené elementy a ich atribúty.

```
<joint name="joint1" type="revolute">
    <origin xyz="0 0 0.25087" rpy="0 0 0" />
    <parent link="base_link" />
    <child link="link1" />
    <axis xyz="0 0 1" />
    <limit lower="-2.9671" upper="2.9671"
           effort="206.26" velocity="4.3633" />
</joint>
```

Výpis 2.5: Príklad štruktúry elementu `<joint>`.

Význam všetkých elementov a atribútov, ktoré môže obsahovať element `<joint>` je nasledovný:

- `name` - atribút názov kĺbu
- `type` - atribút typ kĺbu, existujú rôzne typy: otočný, lineárny, priebežný, prizmatický a pevný
- `<origin>` - posun a rotácia z rodičovského kĺbu do potomka, kĺb sa nachádza v počiatku potomka
- `<parent>` - atribút `link`, názov rodičovského segmentu
- `<child>` - atribút `link`, názov potomka (segmentu)
- `<axis>` - atribút `xyz`, určuje os otáčania kĺbu
- `<calibration>` - referenčná poloha pre kalibráciu absolútnej pozície kĺbu atribút `link`, názov rodičovského segmentu

  `rising` - atribút, pri rotácii v kladnom smere dôjde k vyvolaniu referencie pozície pre nástupnú hranu

  `falling` - atribút, pri rotácii v zápornom smere dôjde k vyvolaniu referencie pozície pre zostupnú hranu
- `<dynamics>` - fyzické vlastnosti kĺbu, užitočné pre simuláciu

  `damping` - atribút, tlmenie

  `friction` - atribút, trenie
- `<limit>` - rozsah pohybu, udáva sa pri otočných a prizmatických kĺboch

  `lower` - limit natočenia pri rotácii kĺbu v zápornom smere od nulového bodu [rad]

  `upper` - limit natočenia pri rotácii kĺbu v kladnom smere od nulového bodu [rad]

  `effort` - maximálny krútiaci moment, ktorý dokáže kĺb vyvinúť [Nm]

  `velocity` - maximálna rýchlosť otáčania, ktorú vie kĺb vyvinúť [rad/s]
- `<mimic>` - kĺb zrkadlí polohu iného kĺbu, hodnotu natočenia je možné upraviť pomocou atribútov

  `multiplier` - atribút, hodnota natočenia kĺbu je získaná vynásobením hodnoty zrkadleného kĺbu

  `offset` - atribút, hodnota natočenia kĺbu je posunutá o konštantu voči hodnote zrkadleného kĺbu
- `<safety_controller>` - bezpečnostné softwarové obmedzenie parametrov kĺbu

  `soft_lower_limit` - limit natočenia pri rotácii kĺbu v zápornom smere od nulového bodu, kedy zasiahne safety_lcontroller [rad]

  `soft_upper_limit` - limit natočenia pri rotácii kĺbu v kladnom smere od nulového bodu, kedy zasiahne safety_controller [rad]

  `k_position` - atribút špecifikujúci závislosť pozície a rýchlosti pohybu kĺbu

  `k_velocity` - atribút špecifikujúci závislosť vyvinutej sily a rýchlosti pohybu kĺbu

2.1.8 Výsledný popis robotického ramena v URDF

Pri vytváraní URDF popisu robotického ramena zapisujeme parametre robotického ramena formou vyššie spomínaných elementov v XML. AK pre tvorbu popisu využijeme nástroj SolidWorks to URDF Exporter 2.1.5, časť týchto potrebných parametrov doplní samotný nástroj, zvyšné je potrebné doplniť ručne. Parametre, ktoré obsahuje náš výsledný popis si priblížime bližšie v nasledujúcej časti textu.

Vlastnosti zotrvačnosti `<inertial>`

Vlastnosti zotrvačnosti sú určené pre každý segment robotického ramena. V prípade vlastností máme definovanú pozíciu ťažiska, hmotnosť segmentu a hodnoty zotrvačnosti.

Ťažisko `<origin>`

Pozíciu ťažiska telesa v priestore je možné získať manuálne výpočtom alebo si ťažisko nechať vypočítať pomocou nástrojov obsiahnutých v CAD softwaroch. Polohu ťažiska je možné vypočítať integrovaním cez objem tuhého telesa, pomocou nasledujúcich vzorcov, prevzaté z [7]:

$$x^* = \frac{\int_V x\rho dV}{\int_V \rho dV} = \frac{1}{m}\int_V x\rho dV \tag{2.2}$$

$$y^* = \frac{\int_V y\rho dV}{\int_V \rho dV} = \frac{1}{m}\int_V y\rho dV \tag{2.3}$$

$$z^* = \frac{\int_V z\rho dV}{\int_V \rho dV} = \frac{1}{m}\int_V z\rho dV \tag{2.4}$$

V našom prípade bola hodnota ťažiska vypočítaná za pomoci Solidworks a pri exporte ju nebolo potrebné dopĺňať.

```
<origin xyz="0.0013701 -6.4154E-05 0.12542" rpy="0 0 0" />
```

Výpis 2.6: Ťažisko základne definované v URDF.

Hmotnosť segmentu`<mass>`

Týmto parametrom sme sa už zaoberali skôr 2.1.4, keď sme v 3D modeli nastavovali hmotnosti segmentov. Hmotnosti sme nastavovali pomocou špecifikovania hustoty materiálu. Vďaka tomuto postupu sme docielili zapísanie tejto hodnoty do modelu, vďaka čomu tieto hodnoty môžu byť následne automaticky doplnené počas exportu do URDF popisu. Taktiež sme týmto umožnili aj správny výpočet ťažiska 2.1.8 a hodnôt zotrvačnosti 2.1.8.

```
<mass value="16.301" />
```

Výpis 2.7: Hmotnosť základne definovaná v URDF.

Zotrvačnosť `<inertia>`

Hodnota zotrvačnosti sa zapisuje pomocou matice zotrvačnosti. Maticu zotrvačnosti je možné odvodiť pomocou nasledujúceho postupu, prevzaté z [17]:

Pre odvodenie vychádzame z kinetickej energie E_k vo vektorovom tvare:

$$E_k = \int_m \frac{1}{2} v^2 dm \tag{2.5}$$

Obvodová rýchlosť v je v tomto prípade zapísaná vo forme trojzložkového vektora, ktorý je možné rozpísať do nasledujúcich relácií:

$$v = \omega \times \rho = \Omega\rho = \hat{\omega}\rho = -\hat{\rho}\omega = -\rho \times \omega \tag{2.6}$$

kde ω je vektor uhlovej rýchlosti, ρ polohový vektor, $\hat{\omega}$, alebo tiež Ω sú antisymetrické matice, priradene k vektoru ω a $\hat{\rho}$ je antisymetrická matica, priradená k vektoru ρ:

$$\omega = \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} \tag{2.7}$$

$$\rho = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{2.8}$$

$$\hat{\omega} = \Omega = \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix} \tag{2.9}$$

$$\hat{\boldsymbol{\rho}} = \begin{bmatrix} 0 & -z & y \\ z & 0 & -x \\ -y & x & 0 \end{bmatrix} \tag{2.10}$$

potom platia relácie:

$$\hat{\omega}^T = \Omega^T = -\hat{\omega} = -\Omega \tag{2.11}$$

$$\hat{\rho}^T = -\hat{\rho} \tag{2.12}$$

pred dosadením do vzťahu 2.5 je potrebné vyjadriť rýchlosť v^2:

$$v^2 = vv = v^T v \tag{2.13}$$

po dosadení do vzťahu 2.5 potom:

$$\begin{aligned} E_k &= \int_m \frac{1}{2} v^2 dm = \int_m \frac{1}{2} (\hat{\omega}\rho)^T (\hat{\omega}\rho) dm = \int_m \frac{1}{2} (\Omega\rho)^T (\Omega\rho) dm = \\ &= \int_m \frac{1}{2} (-\hat{\rho}\omega)^T (\hat{\rho}\omega) dm = \int_m \frac{1}{2} \omega^T \hat{\rho}^T \hat{\rho}\omega dm = \\ &= \frac{1}{2} \omega^T \left(\int_m \hat{\rho}^T \hat{\rho} dm \right) \omega = \frac{1}{2} \omega^T \left(\int_m -\hat{\rho}\hat{\rho} dm \right) \omega = \\ &= \frac{1}{2} \omega^T \left(\int_m -\hat{\rho}^2 dm \right) \omega = \frac{1}{2} \omega^T I_S \omega \end{aligned} \tag{2.14}$$

Maticu zotrvačnosti I_S, ktorej presnú podobu môžeme získať roznásobením:

$$
\begin{aligned}
I_S &= -\int_m \begin{bmatrix} 0 & -z & y \\ z & 0 & -x \\ -y & x & 0 \end{bmatrix}^2 dm = \\
&= -\int_m \begin{bmatrix} 0 & -z & y \\ z & 0 & -x \\ -y & x & 0 \end{bmatrix} \begin{bmatrix} 0 & -z & y \\ z & 0 & -x \\ -y & x & 0 \end{bmatrix} dm = \\
&= \int_m \begin{bmatrix} y^2+z^2 & -xy & -xz \\ -yx & x^2+y^2 & -yz \\ -zx & -zy & x^2+y^2 \end{bmatrix} dm
\end{aligned} \tag{2.15}
$$

$$
I_S = \begin{bmatrix} I_x & -D_{xy} & -D_{xz} \\ -D_{yx} & I_y & -D_{yz} \\ -D_{zx} & -D_{zy} & I_z \end{bmatrix} \tag{2.16}
$$

Výsledná matica zotrvačnosti 2.16 je symetrická a má tvar 3x3. Vďaka tomu je pre vyjadrenie zotrvačnosti potrebných len šesť hodnôt, ktoré ležia na diagonále a pod/nad ňou. Pre výpočet zotrvačnosti známych geometrických telies tak stačí dosadiť podľa predpisu do tejto matice. Pri výpočte zotrvačnosti ramena to však nie je možné, pretože segmenty nášho ramena sú nepravidelných tvarov, pre ktoré neexistujú predpisy pre výpočet. Preto sme v našom prípade využili hodnoty vypočítané pomocou nástrojov v Solidworks. Tieto hodnoty boli následne automaticky vložené do URDF popisu robotického ramena.

```
<inertia ixx="0.16272" ixy="-1.3928E-05" ixz="0.00036162"
         iyy="0.15649" iyz="-0.00011637" izz="0.11345" />
```

Výpis 2.8: Hodnoty zotrvačnosti základne zapísané v URDF.

Tvar segmentu `<link geometry>`

Tvar segmentu je možné definovať niekoľkými spôsobmi. Na výber je niekoľko základných geometrických telies alebo je možné tvar segmentu definovať pomocou súboru, ktorý obsahuje 3D model telesa vo formáte STL. Ak by sme volili jednoduché teleso, mali by sme na vyber z kvádra, gule alebo valca, ktoré by sme definovali pomocou ich rozmerov. V našom prípade sú tvary robotického ramena zložité, pre popis tvaru využívame 3D model segmentov. Do URDF popisu sa takýto model vkladá pomocou elementu `<mesh>`, v ktorom špecifikujeme cestu k súboru pomocou atribútu `filename`. Pri použití nástroja SolidWorks to URDF, je automaticky vložený tento element a taktiež je vyexportovaný aj potrebný súbor STL.

```
<geometry>
   <mesh filename="package://melfa_rv-6sl/meshes/base_link.STL" />
</geometry>
```

Výpis 2.9: Tvar segmentu základne zapísaný v URDF.

Definícia kĺbu `<joint>`

Kĺb sa vždy nachádza medzi dvoma segmentami robotického ramena. Pri definícii kĺbu je potrebné definovať názov pre potreby identifikácia a typ. Typ kĺbu určuje, aký typ pohybu

môže kĺb vykonávať. V URDF je možné definovať rôzne typy, ktoré reprezentujú otáčavý alebo lineárny pohyb.

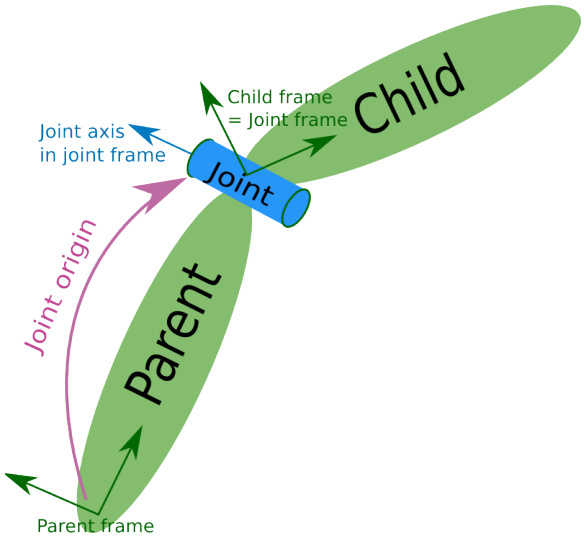
Existuje však výnimka. Je možné definovať aj pevný kĺb, ktorý nie je pohyblivý a slúži pre spájanie segmentov, ktoré je potrebné spojiť napevno.

```
<geometry>
    <joint name="joint1" type="revolute">
</geometry>
```

Výpis 2.10: Definícia kĺbu v URDF.

Pozícia kĺbu `origin`

Pozícia kĺbu v priestore je definovaná pomocou bodu v priestore a natočenia potomka voči rodičovskému kĺbu. Kĺb leží v počiatočnom bode potomka, viď 2.17.



Obr. 2.17: Závislosť kĺbu a segmentov, prevzaté z [37]

```
<origin xyz="0 0 0.25087" rpy="0 0 0" />
```

Výpis 2.11: Pozícia kĺbu `joint1` .

Rodičovský segment kĺbu `parent link`

Element `parent link` obsahuje informáciu o názve rodičovského segmentu kĺbu, viď 2.17.

```
<parent link="base_link" />
```

Výpis 2.12: Definícia rodičovského segmentu pre kĺb `joint1`.

Potomok (nadväzujúci segment) `child link`

Element `child link` obsahuje informáciu o názve nadväzujúceho segmentu, potomka kĺbu, viď obrázok 2.17.

```
<child link="link1" />
```

Výpis 2.13: Definícia potomka pre kĺb `joint1`.

Os otáčania kĺbu `axis`

Element `axis` definuje os v priestore, s ktorou je paralelná os otáčania kĺbu.

```
<axis xyz="0 0 1" />
```

Výpis 2.14: Definícia orientácie osi otáčania kĺbu.

Limity kĺbu `<limit>`

Pri niektorých typoch kĺbov je potrebné špecifikovať limity pohybu kĺbu. Konkrétne je tento parameter potrebné špecifikovať pri typoch `prismatic` a `revolute`. Naše robotické rameno obsahuje kĺby práve týchto dvoch typov. Konkrétne je potrebné definovať maximálny rozsah pohybu, maximálnu silu a rýchlosť, ktorú je kĺb schopný vyvinúť.

Rozsah pohybu určujú dva parametre, spodná (`lower`) a horná (`upper`) hranica rozsahu natočenia. Tieto hodnoty sa udávajú v radiánoch a v prípade nášho robotického ramena sa tieto parametre nachádzajú v užívateľskom manuáli [3]. Manuál však tieto hodnoty udáva v stupňoch, preto je potrebný ich prepočet na radiány pomocou vzťahu 2.17.

$$1^\circ = \pi/180^\circ = 0.005555556\pi = 0.01745329252rad \tag{2.17}$$

Maximálna rýchlosť pohybu (`velocity`), otáčania kĺbu sa udáva v radiánoch za sekundu. Opäť ako v prípade hraníc rozsahu pohybu, sú tieto hodnoty definované v manuáli [3]. V tomto prípade je opäť nutný prepočet podľa vzťahu 2.18, nakoľko výrobca tento parameter udáva v stupňoch za sekundu.

$$1^\circ/s = 0.0174532925rad/s \tag{2.18}$$

Maximálnu silu, krútiaci moment, ktorý dokáže kĺb vyvinúť, udávame pomocou atribútu `effort` v Nm (newton meter). V prípade tohto parametru sa však nedozvieme v manuáli [3] hodnoty pre všetky kĺby, preto je potrebné tieto hodnoty dopočítať. Rovnako ako v prípade výpočtu hmotnosti segmentov, nám pri tomto pomôže servisný portál [23].

Pomocou servisného portálu je možné zistiť maximálne otáčky motorov, ktoré sú 3000 otáčok za minútu pri všetkých šiestich motoroch v našom rameni. Výkony motorov sú následne udané na výrobnom štítku ramena. Z týchto údajov je následne možné pomocou vzťahu 2.21 vypočítať maximálny krútiaci moment, ktorý dokáže servomotor vyvinúť na výstupe. Predtým je však potrebné previesť počet otáčok za minútu (RPM) na počet otáčok za sekundu (Hz), alebo tiež frekvenciu, pomocou vzťahu 2.19.

$$1RPM = 0.01666667Hz \tag{2.19}$$

$$3000RPM = 50Hz \tag{2.20}$$

$$P = \omega M \tag{2.21}$$

vo vzťahu 2.21 je možné rozpísať uhlovú rýchlosť ω:

$$P = \frac{2\pi M f}{60} \tag{2.22}$$

následne si z tohto vzťahu vyjadríme krútiaci moment motoru M, prevzaté z [6] a [49]:

$$M = \frac{60P}{2\pi f} \tag{2.23}$$

Vo výslednom vzťahu nám figurujú výkon P vo *Wattoch* a frekvencia f v *Hz*, ktorú už poznáme z 2.20

Následne je potrebné zistiť prevodový pomer prevodovky. Ten je možné odvodiť na základe pomeru vstupnej a výstupnej rýchlosti. Na vstupe má naše rameno servomotor, ktorého maximálne otáčky poznáme. Taktiež poznáme maximálnu rýchlosť otáčania kĺbu. Táto rýchlosť zodpovedá maximálnej rýchlosti na výstupe prevodovky pri maximálnych otáčkach servomotora. Pre potreby výpočtu je potrebné previesť maximálne otáčky motora z počtu otáčok za minútu, na počet stupňov za sekundu pomocou vzťahu 2.24, nakoľko výstupnú rýchlosť prevodovky máme udávanú práve v tejto jednotke. Výsledný prevodový pomer potom určíme pomocou vzťahu 2.25.

$$1RPM = 0.1666667^{\circ}/s \tag{2.24}$$

$$p = \frac{v_i}{v_o} \tag{2.25}$$

Po získaní prevodového pomeru p, pomocou vstupnej rýchlosti v_i a výstupnej rýchlosti v_o. Výsledný krútiaci moment na výstupe prevodovky M_p získame pomocou prevodového pomeru p a vstupného krútiaceho momentu od motora M_m podľa vzťahu 2.26.

$$M_p = pM_m \tag{2.26}$$

Vypočítaním maximálneho krútiaceho momentu sme zistili posledný dôležitý parameter udávajúci limity otočného alebo pragmatického kĺbu.

```
<limit lower="-2.9671" upper="2.9671"effort="206.26" velocity="4.3633" />
```

Výpis 2.15: Hodnoty definujúce limity otočného kĺbu v URDF.

Kapitola 3

ROS

Obsahom tejto kapitoly je popis a vysvetlenie jednotlivých knižníc a súčastí frameworku ROS [29] a spôsob akým boli využité pri vypracovaní tejto bakalárskej práce.

3.1 ROS – Robot Operating System

ROS, vo voľnom preklade tiež robotický operačný systém, je framework, sada knižníc, konvencií a nástrojov, vytvorených pre vytváranie softwaru pre roboty. Cieľom je zabezpečiť zjednodušenie vývoja softwaru, ktorého úlohou je riadenie robotických zariadení naprieč rôznymi typmi konštrukcií a zameraní, pričom je kladený veľký dôraz na kvalitu a komplexnosť ich chovania. Samotný názov tohto frameworku zvádza k myšlienke, že sa jedná o samostatný operačný systém, no nie je tomu tak. ROS framework je v skutočnosti meta-operačny systém, ktorý pomocou knižníc, ktoré obsahuje, efektívne zabezpečuje komunikáciu operačného systému nad ktorým pracuje s hardwarom robotickej platformy.

V súčasnosti sa ROS teší relatívne veľkej obľube, nakoľko sa jedná open-source framework, ktorý bol od základu navrhnutý tak, aby dokázal pokryť potreby pri vývoji softwaru pre kolaboratívnu robotiku. Láka tak veľké množstvo amatérskych nadšencov robotiky, no predovšetkým mu je venovaná veľká pozornosť zo strany rôznych výskumných skupín prevažne z akademických kruhov. Vďaka svojej architektúre tak poskytuje možnosti využitia nového hardwaru, ale aj softwaru, na základe čoho vznikajú rôzne projekty, napríklad na využitie strojového videnia alebo integrácie umelej inteligencie do robotických aplikácií.

Pre potreby tejto práce je využitá stabilná verzia frameworku ROS s označením Melodic [35], ktorá je určená pre inštaláciu nad operačným systémom Ubuntu vo verzii 18.04. V súčasnosti je dostupná aj stabilná verzia s označením Noetic[41] určená pre operačný systém Ubuntu 20.04. Pri tejto verzii prebieha jej aktívny vývoj a jedná sa o relatívne nedávno vydanú verziu. Staršiu verziu sme zvolili z dôvodu jej ukončeného vývoja a stabilnej verzie, ako aj faktu, že pre potreby práce využívame aj knižnicu, ktorá bola špeciálne vytvorená pre robotické ramená Mela ešte vo verzii Kinetic [34] (Ubuntu 16.04) a rozdiel medzi týmito verziami je menší, ako v prípade verzie Kinetic (Ubuntu 16.04) a Noetic (Ubuntu 20.04), čo by v prípade problémov mohlo viesť k ich ľahšiemu riešeniu, nakoľko veľká časť problémov, ktoré môžu nastať vo verzii Melodic sú do veľkej miery pokryté funkčnými riešeniami zo strany komunity.

3.2 ROS - základné koncepty

V prípade ROS delíme koncepty na tri základné úrovne/skupiny , ktoré pokrývajú určité časti princípov a funkcionality ROS ako celku. Nasledujúce informácie boli prevzaté z [32].

3.2.1 ROS - úroveň súborového systému

Úroveň súborového systému, označovaná v ROS ako *Filesystem level*, pokrýva predovšetkým oblasť súborov, ktoré sú uložené na disku. Do tejto kategórie spadajú:

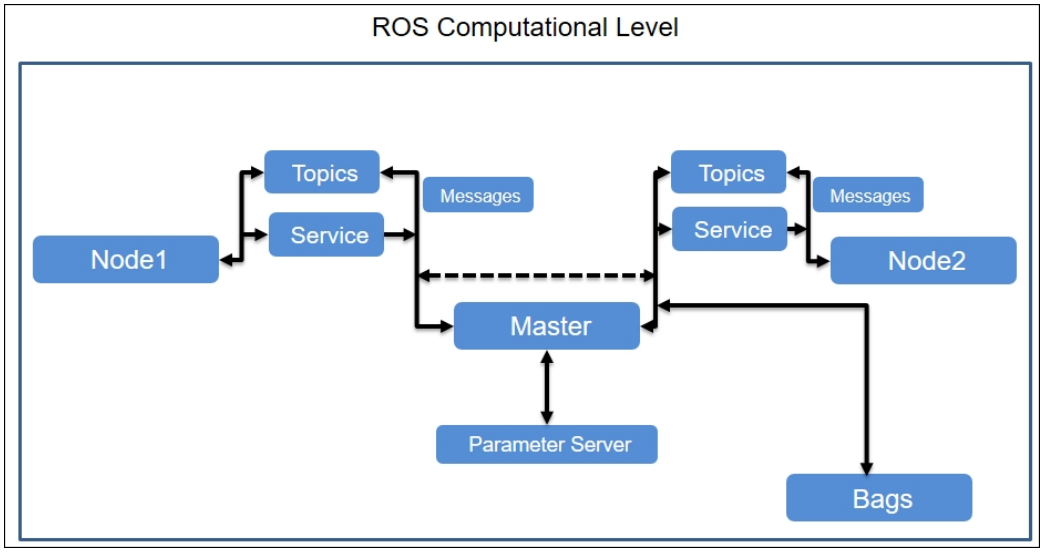
- **Balíčky (Packages)** - organizácia softwaru v ROS je postavená na tomto type organizačných jednotiek, ktoré môžu obsahovať procesy behu systému, nazývané tiež uzly (nodes) 3.2.2, ale aj iné podstatné komponenty ako knižnice, datasety, konfiguračné súbory a mnohé ďalšie komponenty, ktoré je z hľadiska organizácie vhodné udržiavať v spoločnom balíčku komponentov. Balíčky sú tiež najzákladnejšou jednotkou, ktorú je možné skonštruovať a distribuovať.
- **Špecializované balíčky (Metapackages)** - skupina balíčkov ktorá je na sebe závislá, využívajú sa predovšetkým pre zabezpečenie spätnej kompatibility.
- **Manifest k balíčku (Package Manifest)** - súbor vo formáte XML, poskytujúci metadáta o konkrétnom balíčku.
- **Repozitáre (Repositories)** - kolekcia balíčkov, ktorá zdieľa bežný veršovací systém.
- **Správy (Message (msg) types)** - súbory popisujúce formát a dátovú štruktúru správ zasielaných v rámci ROS.
- **Služby (Service (srv) types)** - súbory popisujúce formát a dátovú štruktúru komunikácie typu otázka a odpoveď, pomocou ktorej komunikujú jednotlivé služby v ROS.

3.2.2 ROS - výpočtová úroveň

Výpočtová úroveň, označovaná v ROS ako *Computation Graph Level*, je peer-to-peer sieť procesov v ROS, ktoré spoločne spracovávajú dáta. Táto sieť sa skladá z niekoľkých častí:

- **Uzly (Nodes)** - sú procesy, ktoré zabezpečujú výpočty. Vďaka modularite, ktorú ROS poskytuje, riadenie robota obsahuje veľké množstvo takýchto uzlov, z ktorých každý disponuje určitou špecifickou funkcionalitou, ktorú v množine zastáva.
- **Hlavný uzol (Master)** - špeciálny uzol, ktorý spravuje ostatné uzly, riadi registráciu uzlov do výpočtovej siete a upozorňuje ostatné uzly v sieti na aktuálne zmeny, riadi tok správ zasielaných medzi uzlami a zabezpečuje spúšťanie potrebných služieb. Bez hlavného uzla by ostatné uzly neboli schopné vzájomnej komunikácie a nemali by o sebe vedomosť.
- **Server pre ukladanie parametrov (Parameter Server)** - umožňuje v sebe ukladať dáta vo forme kľúč – hodnota. V súčasnosti je už súčasťou hlavného uzlu.
- **Správy (Messages)** - uzly komunikujú medzi sebou formou správ, jednoduchých dátových štruktúr nesúcich potrebné dáta.

- **Témy (Topics)** - správy sú v systéme ROS smerované pomocou schémy, kedy uzly publikujú alebo odoberajú určitý typ správ. Názov témy určuje typ/obsah zasielaných správ. Tému si tak vieme predstaviť ako silne typovanú zbernicu pre zasielanie dát, do ktorej časť uzlov správy zasiela a iné uzly ich následne prijímajú, pokiaľ sú tieto správy správneho typu. Jednotlivé uzly o svojej existencii v rámci témy nevedia. Taktiež každý uzol môže byť prihlásený k publikovaniu alebo odberu viacerých tém.
- **Služby (Services)** – na rozdiel od správ, ktoré sú jednosmerným typom komunikácie, služby zabezpečujú obojsmernú komunikáciu typu otázka a odpoveď medzi uzlami, ktorá je často využívaným typom v distribuovaných systémoch.
- **Bag** - je formát využívaný pre ukladanie správ zasielaných v systéme, pre možnosť spätného prehratia počas testovania. Prevažne sa takto ukladajú dáta zo senzorov, ktoré je ťažké zozbierať, pripadne pri testovaní takto zachytiť určité nežiadané stavy, ktoré môžu byť spätne analyzované.



Obr. 3.1: ROS Computation Graph Level, prevzaté z [19]

3.2.3 ROS - komunitná úroveň

Komunitná úroveň, označovaná v ROS ako Community Level, zabezpečuje zdieľanie zdrojov, softwaru a vedomostí naprieč komunitou. Táto úroveň pokrýva oblasti:

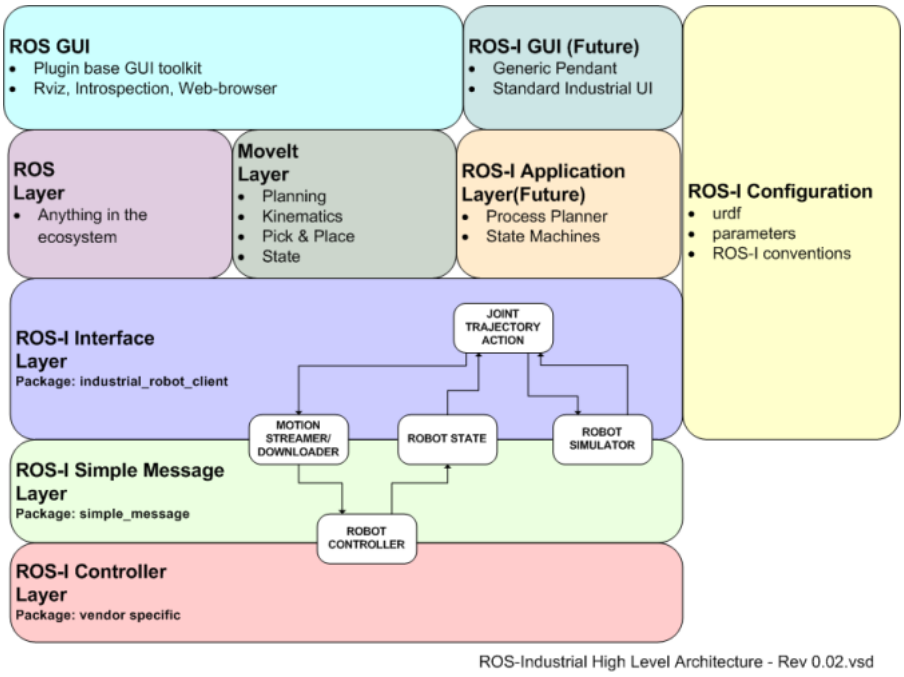
- **Inštalačné balíky (Distributions)** - kolekcie verziovaných balíčkov určených pre inštaláciu, ktoré zohrávajú rovnakú úlohu ako inštalačné balíky jednotlivých verzií operačného systému Linux
- **Repozitáre (Repositories)** - ROS je postavený na princípe zdieľania kódu v sieti repozitárov, do ktorej môžu rôzne inštitúcie prispievať vlastnými softwarovými komponentami, ktoré vyvíjajú pre vlastné roboty
- **The ROS Wiki** - wiki stránky tvorené komunitou ROS, ktoré sú hlavným zdrojom informácií a dokumentácie o frameworku ROS.

- **Systém nahlasovania chýb (Bug Ticket System)** - systém pre nahlasovanie objavených chýb v balíčkoch
- **Zoznamy mailových adries (Mailing Lists)** - tieto zoznamy sú určené pre zasielanie správ a komunikáciu medzi užívateľmi ROS a šírenie informácií o updatoch, ako aj komunikačný kanál pre kladenie otázok ohľadom ROS
- **ROS odpovede (ROS Answers)** - fórum určené pre pokladanie otázok, na ktoré môžu členovia komunity odpovedať.
- **Blog** - ros.org blog poskytuje pravidelne novinky zo sveta ROS, vrátane fotografií a videí.

3.2.4 ROS - Industrial

ROS Industrial [39] je open-source projekt, ktorý vznikol v roku 2012 [18] ako odnož pôvodného ROS. Cieľom ROS Industrial je priniesť do priemyselnej robotiky a automatizácie nové technológie a možnosti za pomoci pokročilého robotického softwaru. Napriek veľkému potenciálu a širokým možnostiam sa zatiaľ tento systém výraznejšie v priemyselnej oblasti nepresadil.

ROS - Industrial poskytuje stabilnú verziu ROS, ktorá spĺňa všetky požiadavky potrebné pre nasadenie v priemyselnom prostredí. Jedná sa tak o komplexný balík knižníc vychádzajúcich z pôvodného ROS, ktoré sú potrebné pre zaistenie plnej funkcionality a zároveň splnenie všetkých, najmä bezpečnostných štandardov vyžadovaných v priemysle. Nespornou výhodou je tiež fakt, že ROS - Industrial je možné relatívne jednoducho rozšíriť o nový software vyvinutý výskumnými skupinami. Medzi výhody tiež patrí možnosť voľne upravovať kód podľa potreby a dostupnosť open-source ovládačov, ktoré nie sú závislé na výrobcoch hardwaru.



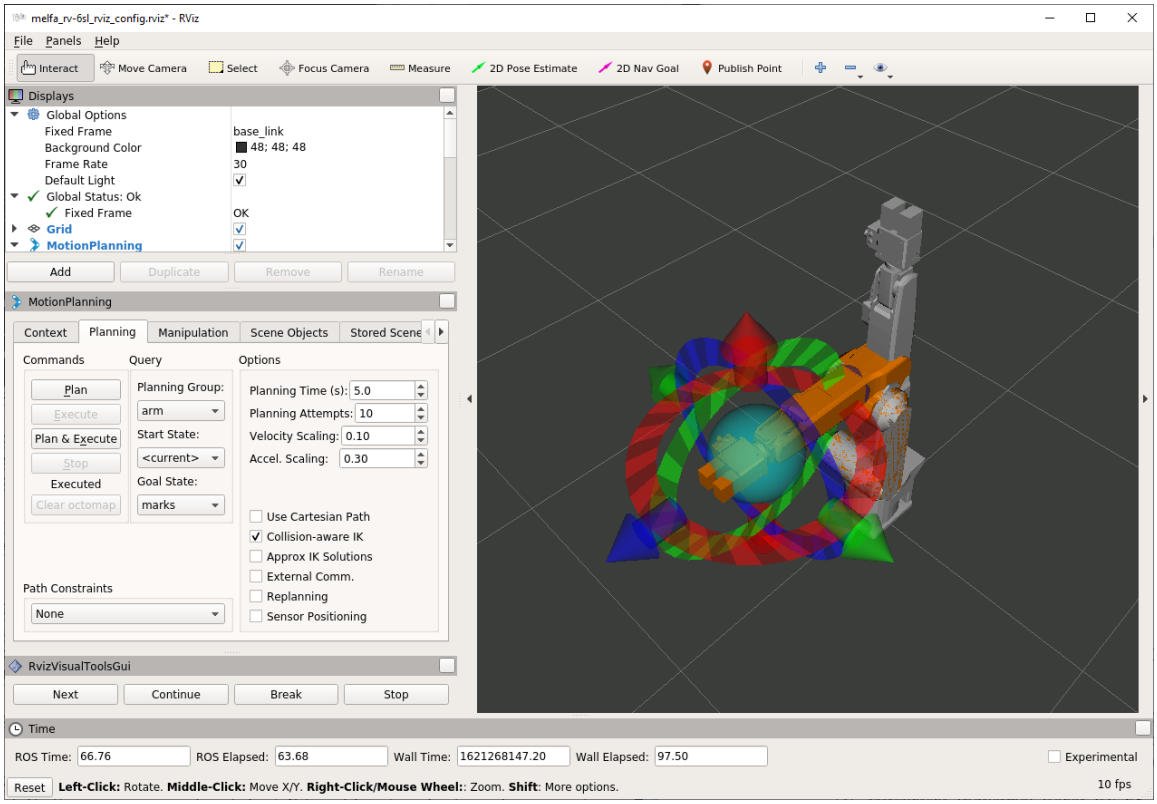
Obr. 3.2: ROS - Industrial, architektúra , prevzaté z [39]

3.3 RViz

RViz [36] je vizualizačný nástroj, ktorý je súčasťou ROS, o čom napovedá aj samotný názov, ktorý je skratkou ROS visualization. Úlohou tohto nástroja je poskytnutie grafického rozhrania pre potreby zobrazenia aktuálneho stavu systému. Dokážeme tak pomocou neho zobrazovať nie len aktuálny stav samotného robota, ale aj rôzne iné hodnoty, ako napríklad aktuálne hodnoty zo snímačov, či dokonca obrazový výstup z kamier. Pre potreby zobrazenia 3D modelu využíva popis robota vo formáte URDF.

Celú túto funkcionalitu poskytuje vďaka tomu, že sa v skutočnosti jedná o jeden z uzlov ROS, vďaka čomu sa môže prihlásiť k odberu rôznych Topicov (tém), z ktorých následne prijíma správy a vykresľuje ich obsah.

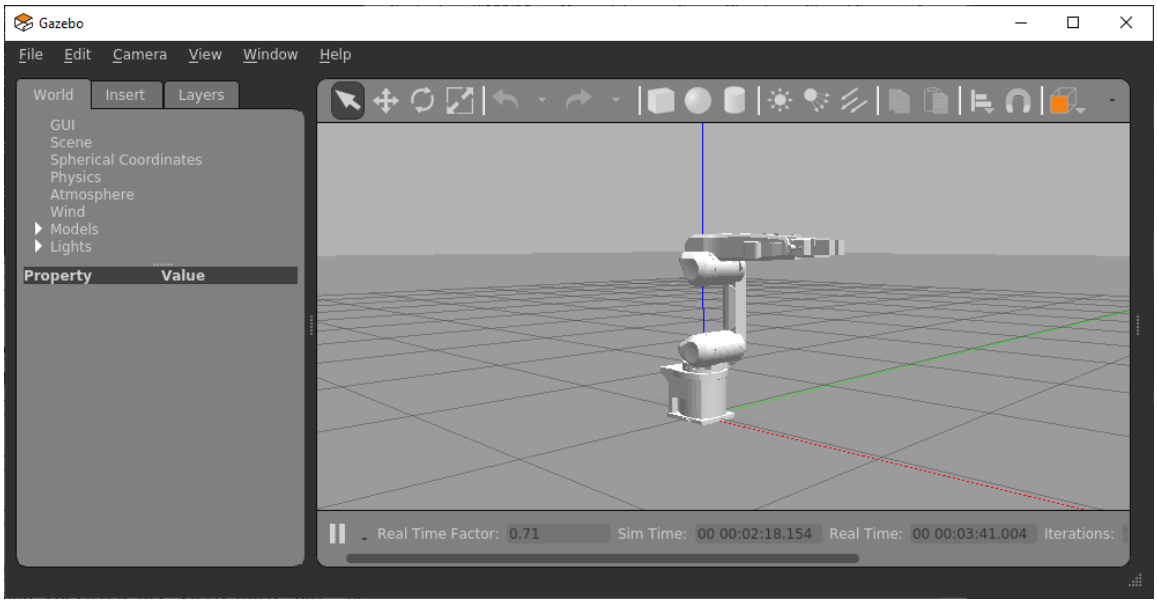
Okrem zobrazovania aktuálneho stavu tiež podporuje komunikáciu s plánovacími knižnicami ako napríklad *MoveIt!* 4. Pomocou grafického rozhrania je tak možné nastaviť cieľovú polohu, ktorá je následne naplánovaná a zobrazená vo vizualizácii.



Obr. 3.3: Vizualizačný nástroj RViz

3.4 Gazebo

Simulačný nástroj Gazebo [11], ktorý vznikol už v roku 2002 na Univerzite v južnej Kalifornii. V roku 2009 bol do simulátora zaintegrovaný ROS. Následne sa veľmi rýchlo stal populárny v komunite a v súčasnosti sa tak využíva ako hlavný simulačný nástroj. Jeho hlavnou úlohou je poskytnutie kvalitného simulačného prostredia pre potreby testovania. Súčasťou simulátoru je niekoľko fyzikálnych modelov. Pre potreby zobrazenia rovnako ako RViz 3.3, využíva popis robota vo formáte URDF. V súčasnosti je tento nástroj súčasťou niektorých inštalačných balíčkov distribuovaných samotným ROS.



Obr. 3.4: Simulačný nástroj Gazebo

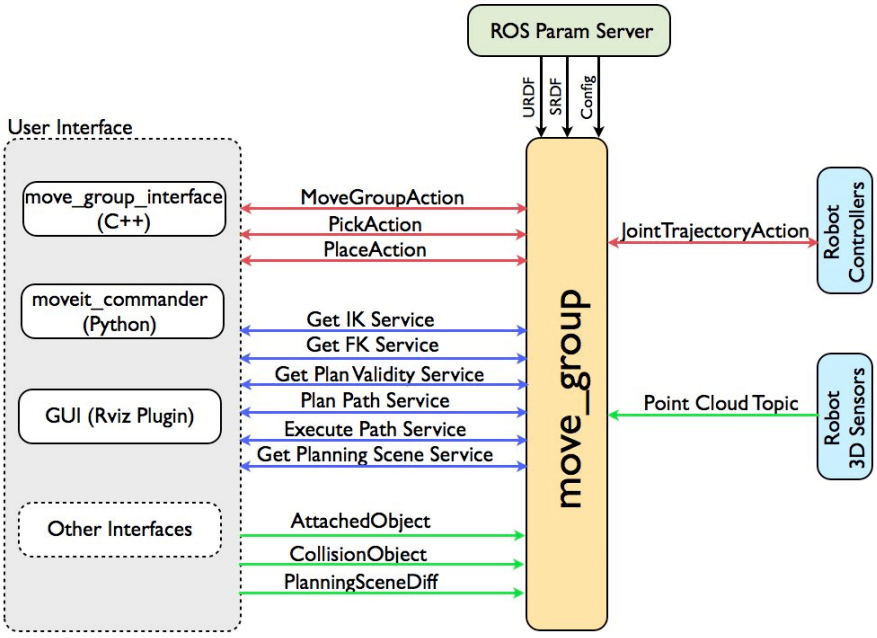
Kapitola 4

MoveIt!

MoveIt! je open-source projekt, na ktorom sa podieľa veľká medzinárodná komunita z niekoľkých organizácií. Výsledkom tohoto projektu je framework, ktorý rozširuje funkcionalitu ROS o možnosti plánovania trajektórie robotov. Od samotného vzniku bol tento framework cielený pre využitie s ROS.

4.1 Architektúra

Architektúra frameworku MoveIt! bola vytvorená tak, že po spustení je vytvorený uzol `move_group`, ktorý umožňuje prihlásenie sa do peer-to-peer siete ROS. Vďaka tomu dokáže MoveIt! komunikovať so zvyškom siete a využívať prostriedky obsiahnuté v ROS. Využíva tak napríklad nástroj RViz pre vizualizáciu trajektórie alebo komunikuje s parametrickým serverom, z ktorého získava potrebné informácie, ako napríklad URDF robota.



Obr. 4.1: Architektúra MoveIt!, prevzaté z [26].

Uzol `move_group` plní úlohu centrálneho prvku architektúry. Okrem komunikácie s ostatnými uzlami plní tiež funkciu plánovača trajektórie robota.

Pre naplánovanie trasy potrebuje tento uzol niekoľko dôležitých parametrov. Tieto parametre získava z parametrického servera, ktorý je obsiahnutý v hlavnom uzle (Master). Z tohto uzla získava okrem URDF popisu robota aj ďalšie potrebné parametre, ako konfiguračný súbor a SRDF (Semantic Robot Decription Format) popis robota, obsahujúci špecifikáciu pohybových skupín robota, základné nastavenia a ďalšie potrebné informácie. Tento formát si bližšie popíšeme v časti venovanej *MoveIt! Setup Assistant* 4.4.

Konfiguračný súbor obsahuje potrebné informácie pre nastavenie tohto uzlu, ako aj jeho jednotlivých funkcií. V prípade funkcie plánovača, tak nastavujeme, aká knižnica plánovacích algoritmov má byť použitá. Samotný plánovač by bez zadania knižnice nedokázal trasu naplánovať. V našom prípade využívame knižnicu **OMPL**, ktorá patrí medzi najčastejšie používané.

4.2 Plánovanie trajektórie

Plánovač podporuje niekoľko spôsobov plánovania trajektórie, ktoré sú závislé na spôsobe definície požadovanej koncovej polohy a ďalších parametrov. Koncovú polohu môžeme definovať buď pomocou natočenia jednotlivých kĺbov, alebo pozície koncového bodu robotického ramena v karteziánskom priestore. Pri zvolení pozície pomocou natočenia kĺbov je trajektória vypočítaná pomocou priamej kinematiky. Ak však zvolíme pozíciu pomocou súradníc koncového bodu, plánovač využíva pre plánovanie inverznú kinematiku, čo predlžuje čas výpočtu trajektórie a nie je tiež zaručené, či dokáže takúto trasu úspešne naplánovať.

V oboch prípadoch, ak sa podarí úspešne naplánovať trajektóriu, je vypočítaná v čo najoptimálnejšej podobe. Znamená to, že táto trasa je plánovaná tak, aby nedochádzalo k zbytočnému pohybu kĺbov. V praxi tak trajektória opísaná nástrojom počas pohybu medzi začiatočným a koncovým bodom nemusí prebiehať po priamke. Toto chovanie je vhodné, ak nám záleží len na výslednej pozícii robotického ramena a nástroja. Ak však naša aplikácia vyžaduje aj definované chovanie nástroja počas presunu medzi počiatočným a koncovým bodom, volíme spôsob plánovania Karteziánskej trasy. V tomto prípade tak koncový bod ramena/nástroj opíše trajektóriu, ktorá bola vypočítaná pomocou interpolácie zadaných bodov. Preto, ak uvažujeme prípad, kedy požadujeme presun ramena po priamke do koncovej polohy, počas ktorej musí nástroj udržiavať určité natočenie v priestore volíme tento typ trajektórie. Pre predstavu môžeme uvažovať prípad, keď je úlohou robota presúvať otvorené nádoby s tekutinou. V prípade optimálnej trajektórie by mohlo dôjsť k nežiadúcemu natočeniu nástroja a vyliatiu tekutiny. Pri Karteziánskej trajektórii by sme zvolili vodorovné natočenie nástroja, ktoré by udržiavalo počas celého presunu nádobu vo vodorovnej polohe a k rozliatiu tekutiny by tak nedošlo.

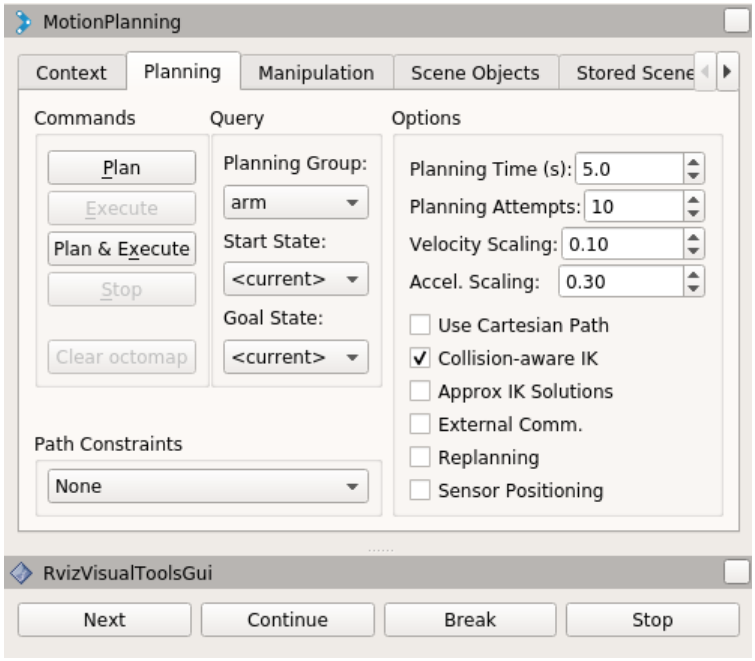
Pri plánovaní trajektórie zohrávajú tiež veľkú úlohu nastavenia pre vyhnutie sa kolízii. Pri optimálnom type trajektórie dokáže plánovač naplánovať s veľkou mierou úspešnosti trajektóriu, pri ktorej sa dokáže vyhnúť prekážkam v priestore. Pri karteziánskom type plánovania je táto možnosť tiež dostupná, no plánovač nie vždy dokáže takúto trajektóriu vytvoriť, nakoľko pri takejto trajektórii má viac obmedzujúcich parametrov.

Úspešnosť plánovania tiež do veľkej miery ovplyvňuje konštrukcia samotného robotického ramena, rozsah pohybu kĺbov, ich počet a tvar segmentov. Preto pri nastavení plánovača môžeme nastaviť aj časový limit pre výpočet trajektórie alebo počet pokusov.

4.3 Užívateľské rozhranie

Pre komunikáciu s uzlom `move_group` využívame zasielanie správ pomocou topicov, ku ktorým je uzol prihlásený. Tieto správy môžeme generovať a prijímať niekoľkými spôsobmi.

Ako sme spomínali už pri popise architektúry 4.1, dokáže uzol `move_group` komunikovať s rozhraním RViz. V prostredí RVIz sú pre túto komunikáciu obsiahnuté pluginy, ktoré po pridaní do panela nástrojov automaticky začínajú komunikovať s uzlom `move_group`. V niektorých prípadoch je potrebné zmeniť nastavenia týchto nástrojov pre odber správneho topicu v rámci komunikácie s naším uzlom. Na výber je pomerne široká ponuka týchto pluginov s rôznou ponukou funkcionality. V našom prípade používame v prostredí RViz dva pluginy - MotionPlanning a RVizVisualToolGUI.



Obr. 4.2: Pluginy MoveIt! v prostredí RViz.

Pri komunikácii s uzlom `move_group` je tiež možné využiť rozhranie, ktoré je naprogramované v jazykoch C++ alebo Python, za pomoci knižníc `move_group_interface` (C++) a `moveit_commander` (Python).

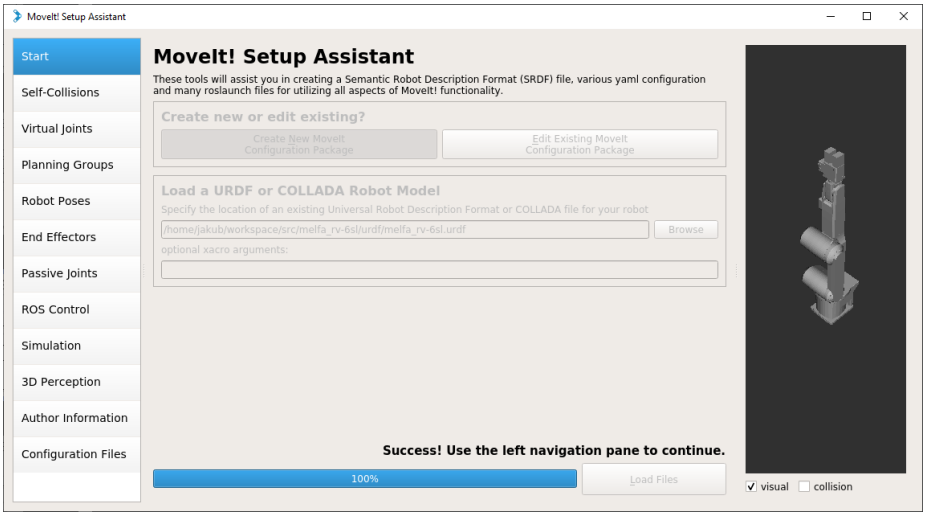
Pre účely tejto bakalárskej práce sme vytvorili jednoduché rozhranie v jazyku C++, ktoré je možné ovládať pomocou terminálu. Pomocou tohto rozhrania je možné plánovať trajektóriu robotického ramena pomocou bodov v karteziánskom priestore s využitím inverznej kinematiky. Motiváciou pre vytvorenie tohto rozhrania bol fakt, že pomocou nástrojov v RViz je možné nastaviť cieľové pozície len na základe natočenia kĺbov, prípadne interaktívnym nastavením pozície vo vizualizačnom okne. Naše rozhranie podporuje plánovanie optimálnej aj karteziánskej trajektórie na základe určenia súradníc požadovaných pozícií. Bola pridaná aj možnosť nastaviť cieľovú polohu ramena pomocou natočenia kĺbov obdobne ako v rozhraní v RViz. Po spustení nášho rozhrania je vytvorený uzol, ktorý po registrácii v ROS môže komunikovať s uzlami MoveIt!. Návod na použitie tohto nástroja sa nachádza v prílohe A.

4.4 MoveIt! Setup Assistant

Nástroj MoveIt! Setup Assistant (MSA) [28], ako už názov napovedá, je určený pre pomoc pri nastavení MoveIt! v rámci nového projektu. Pomocou tohto nástroja si vytvoríme balíček, ktorý bude obsahovať základné komponenty potrebné pre plánovanie trajektórie pomocou plánovača MoveIt!. Súčasťou výsledného balíčka budú potrebné konfiguračné súbory, špecifikácia robotického ramena v SRDF, súbory obsahujúce softwarové kontroléry a spúšťacie súbory.

4.4.1 Vytvorenie balíčku

Pre vytvorenie balíčku pomocou MSA je potrebný popis robotického ramena v URDF. Po úspešnom načítaní tohto popisu do MSA následne v niekoľkých krokoch nastavíme potrebné parametre a vygenerujeme výsledný balíček.



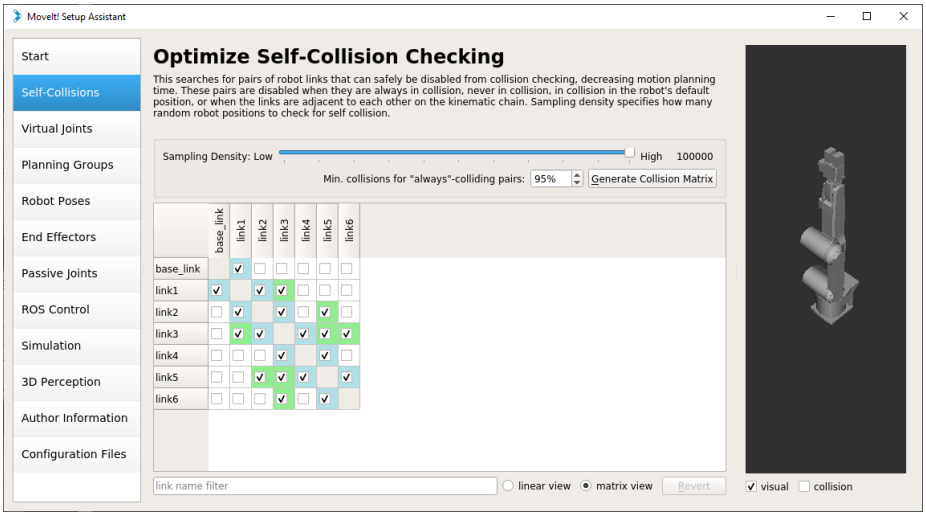
Obr. 4.3: Úvodná obrazovka MSA s načítaným modelom.

Self-Collision

V tomto kroku je vygenerovaná matica, ktorá reprezentuje, medzi ktorými časťami ramena môže dôjsť k vzájomnej kolízii alebo naopak k vzájomnej kolízii vzhľadom na vlastnosti konštrukcie dôjsť nikdy nemôže. Výsledná matica je zapísana v SRDF súbore.

Farby políčok matice reprezentujú stavy, ktoré môžu nastať, viď obrázok 4.4:

- Biela - tieto sa môžu dostať do vzájomnej kolízie
- Modrá - tieto segmenty sa nikdy nedostanú do kolízie, pretože sa jedná o susediace segmenty
- Zelená - tieto segmenty sa nemôžu dostať do vzájomnej kolízie vďaka obmedzeniam z rozsahu pohybu



Obr. 4.4: Matica kolízií vygenerovaná pomocou MSA.

Virtual Joints

V tomto kroku sa vytvárajú virtuálne kĺby. Napríklad sa tu môže pridať virtuálny pevný kĺb, ktorý bude slúžiť k fixácii ramena do podložky v simulácii. V našom prípade tento krok vynecháme z dôvodu, že sme takýto kĺb manuálne už pridali počas prvotného testovania do pôvodného URDF, na základe nápovedy zobrazenej počas behu RViz.

```
<link name="world" />
<joint name="world_joint" type="fixed">
    <parent link="world" />
    <child link="base_link" />
    <origin xyz="0.0 0.0 0.0" rpy="0.0 0.0 0.0" />
</joint>
```

Výpis 4.1: Virtuálny kĺb definovaný v URDF.

Planning Groups

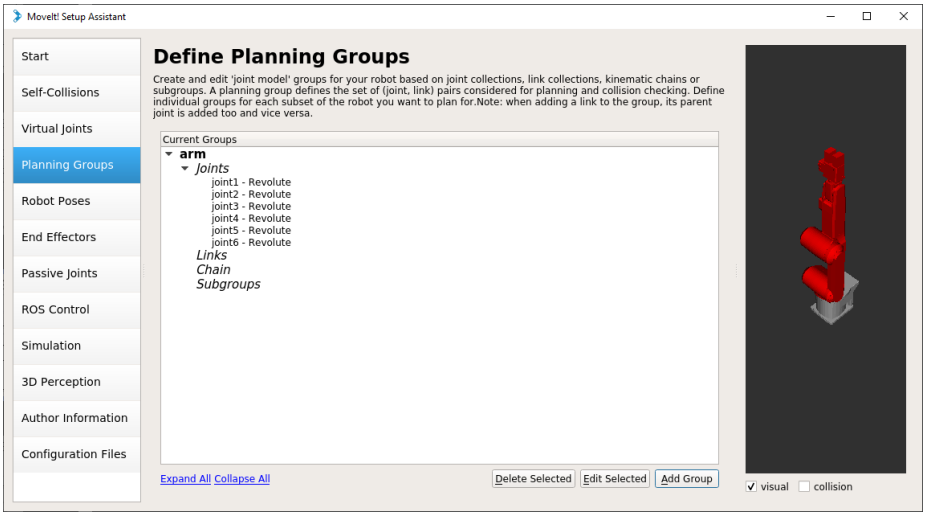
V tomto kroku, ako už samotný názov napovedá, bude potrebné definovať pohybové skupiny robotického ramena. Na základe toho je možné následne plánovať trajektóriu.

Pri vytváraní je nutné vytvoriť minimálne jednu pohybovú skupinu. Viac skupín sa vytvára, ak robot obsahuje viac ramien alebo nástroj s pohyblivými časťami, ktoré by mohli ovplyvniť trajektóriu, napríklad gripper. Pre priemyselné ramená sa preto vytvárajú dve pohybové skupiny. V našom prípade budeme vytvárať len jednu pohybovú skupinu, pretože gripper v našom prípade nie je funkčný, a preto nie je potrebné plánovať jeho otváranie a zatváranie. Pri vytváraní pohybovej skupiny sa pridávajú všetky kĺby ramena okrem virtuálneho kĺbu `world_joint`, ktorý sa využíva len pre potreby simulácie.

Ďalej sa v tomto kroku nastavuje pri výbere kĺbov, pomocou akej knižnice/pluginu bude riešiť kinematiku. V našom prípade sme zvolili odporúčaný *KDLKinemticsPlugin*. Taktiež sa v tomto kroku nastavuje maximálny čas pre beh jedného cyklu riešenia kinematickej úlohy, maximálny počet pokusov a rozlíšenie, s ktorým bude prehľadávaný priestor pri hľadaní riešenia, ak by sme využívali robotické rameno s redundantnými kĺbmi (rameno so

siedmimi a viac kĺbmi). Vo výslednom balíčku sa tieto parametre nachádzajú v súbore *kinematics.yaml*.

Pohybové skupiny sú zapísané v SRDF súbore. V našom prípade sme pohybovú skupinu pomenovali *arm*. Na obrázku 4.5 je možné vidieť pohybovú skupinu, vo vizualizácii je zobrazená červenou farbou.



Obr. 4.5: Definícia pohybovej skupiny *arm* v MSA.

Robot Poses

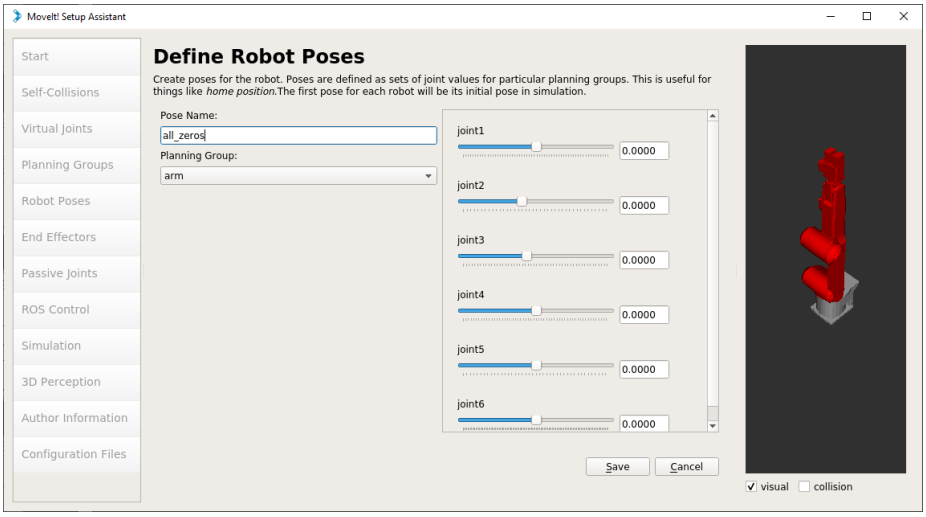
V tomto kroku je možné definovať polohy robotického ramena, ktoré môžu byť užitočné pri simulácii alebo pri práci s reálnym ramenom. Tieto polohy sú súčasťou SRDF súboru.

V našom prípade sme preddefinovali tri polohy:

- **all_zeros** - poloha, kedy sú všetky kĺby v nulových polohách (rameno je vystreté vo vertikálnej polohe), viď obrázok 4.6
- **marks** - poloha, kedy je hodnota natočenia tretieho kĺbu +90°, zvyšné kĺby sú v nulových polohách (rameno má segmenty zarovnané podľa rysiek)
- **folded** - rameno je v polohe, v ktorej sa nachádza pri preprave z výrobného závodu, druhý kĺb je natočený do maximálnej zápornej polohy a tretí kĺb je natočený do maximálnej kladnej polohy, zvyšné kĺby sú v nulových polohách

End Effectors

V tomto kroku sa definujú bližšie vlastnosti pre effektory (nástroje). Effektorom sú priradené plánovacie skupiny zo sekcie Planning Groups 4.4.1, ako aj rodičovský segment, od ktorého sa bude odvodzovať jeho poloha. V našej implementácii effektor nevyužívame a je ako pasívny komponent súčasťou koncového segmentu.



Obr. 4.6: Definícia polohy robotického ramena v MSA.

Passive Joints

V tomto kroku sa definuje, ktoré kĺby sú pasívne - nepotrebujeme poznať ich stav. Pri niektorých robotických ramenách sa využívajú pomocné pasívne segmenty, ktoré slúžia napríklad na spojenie niektorého aktívneho segmentu s protiváhou. Naše robotické rameno žiaden takýto kĺb neobsahuje, preto tento krok vynecháme.

ROS Control

V tomto kroku sa nastavujú potrebné softwerové kontroléry, potrebné pre komunikáciu MoveIt a ROS Control [42]. Tieto kontroléry sú následne uložené v súbore *ros_controllers.yaml*. Tieto kontroléry slúžia ako náhrada za hardwarove kontroléry pri doma vyrobených robotických ramenách. Náš priemyselný robot hardwarovým kontrolérom disponuje, no časť jeho funkcionality preberáme do našej aplikácie. Preto využijeme len minimálnu množinu z dostupných softwarových kontrolérov.

- **JointStateController** - tento kontrolér slúži pre publikovanie aktuálneho stavu robota (reálneho aj simulovaného). Stavy natočenia kĺbov následne publikuje v topicu (téme) `/joint_states`. Tento kontrolér je automaticky pridaný a vygenerovaný MSA.

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

- **FollowJointTrajectory** - tento typ kontroléru ako už názov napovedá, slúži pre sledovanie trajektórie jednotlivých kĺbov robotického ramena.

```
- name: joint_trajectory_controller
  action_ns: follow_joint_trajectory
  default: True
  type: FollowJointTrajectory
```

```
joints:
  - joint1
 ...
  - joint6
```

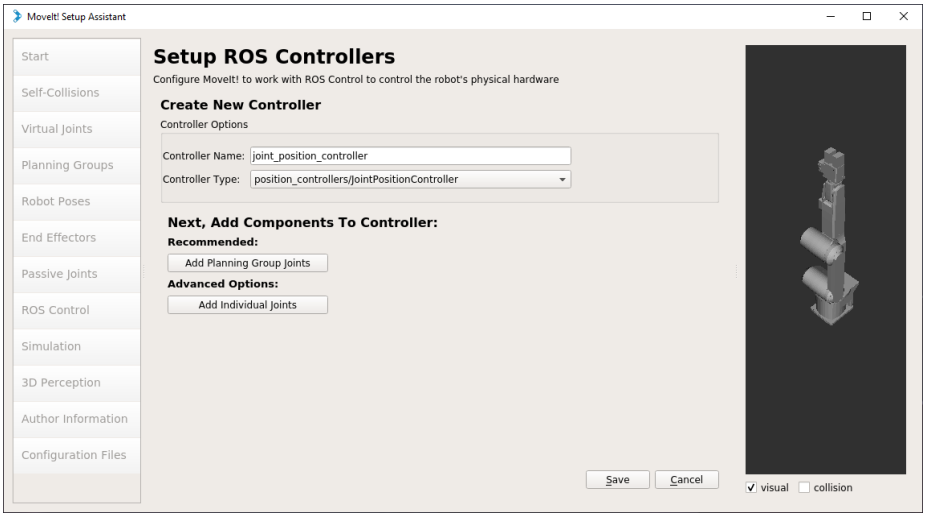
- **JointPositionController** - tento typ kontroléru riadi natočenie jednotlivých kĺbov robotického ramena

```
joint_position_controller:
  type: position_controllers/JointPositionController
  joints:
    - joint1
    ...
    - joint6
  gains:
    joint1:
      p: 100
      d: 1
      i: 1
      i_clamp: 1
    ...
    joint6:
      p: 100
      d: 1
      i: 1
      i_clamp: 1
```

- **JointTrajectoryController** - tento typ kontroléru riadi trajektóriu jednotlivých kĺbov robotického ramena

```
joint_trajectory_controller:
  type: position_controllers/JointTrajectoryController
  joints:
    - joint1
    ...
    - joint6
  gains:
    joint1:
      p: 100
      d: 1
      i: 1
      i_clamp: 1
    ...
    joint6:
      p: 100
      d: 1
      i: 1
      i_clamp: 1
```

Tieto kontroléry boli zvolené na základe kontrolérov použitých v balíčku `melfa_driver` 5.1.1. Pri pridaní týchto kontrolérov MSA je potrebné určit typ kontroléru a ktoré kĺby alebo skupiny kĺbov budú konkrétnemu kontroléru prislúchať, viď obrázok 4.7.



Obr. 4.7: Výsledný zoznam kontrolérov v MSA.

Simulation

V tomto kroku MSA vygeneruje nový obsah URDF popisu robotického ramena, respektíve doplní pôvodnú definíciu o plugin ROS control, virtuálne motory a prevodovky pre potreby simulácie v simulátore Gazebo. Tieto zmeny sú označené zelenou farbou a je potrebné ich aplikovať do pôvodného URDF súboru.

V našom prípade je potrebné vytvoriť nový URDF popis robotického ramena a zachovať aj pôvodný. Typ rozhrania využívaný pre riadenie týchto virtuálnych komponentov je totiž rovnaký ako typ rozhrania využívaný pre riadenie reálneho ramena. Pri spustení aplikácie s°reálnym tak dochádzalo k opakovanému spusteniu tohto rozhrania, čo viedlo k nestabilnému chodu aplikácie a pádom rozhrania. Pri niektorých typoch rozhraní je možné problém riešiť definovaním simulačného módu, co zabezpečí, že sa toto rozhranie spustí len v prípade ak je spustený aj simulátor Gazebo [1][10]. V prípade nami využívaného typu táto možnosť nie je dostupná. Preto pri práci so simulátorom budeme využívať URDF popis obsahujúci tieto virtuálne komponenty a pri práci s reálnym ramenom zase pôvodný URDF popis ramena.

3D Perception

V tomto kroku je možné konfigurovať rôzne snímače a kamerové systémy určené pre orientáciu robota v priestore. V našom prípade žiadne senzory nevyužívame, preto tento krok vynecháme.

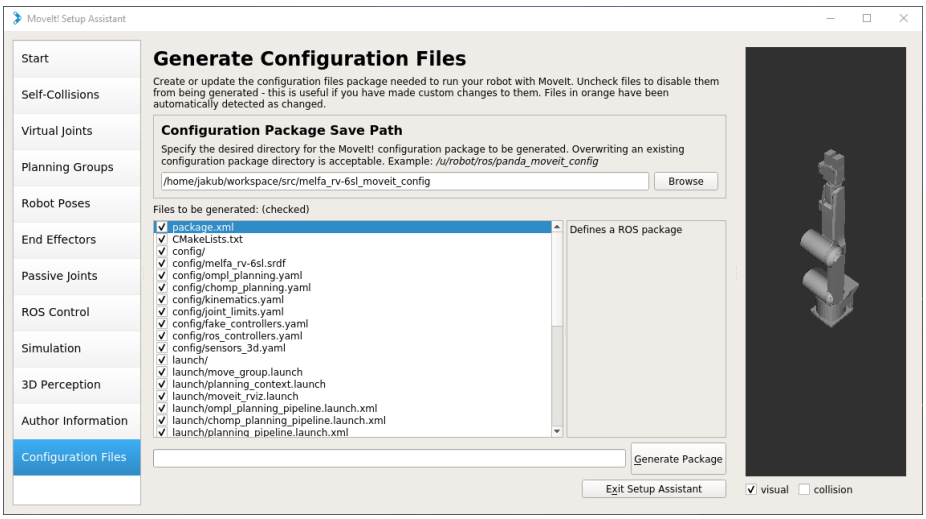
Author Infromation

V tomto kroku sa vypĺňajú informácie o autorovi konfiguračného balíčku.

Obr. 4.8: URDF doplnený o potrebné zmeny pomocou MSA.

Configuration Files

Toto je finálny krok vytvárania konfiguračného balíčku. V tomto kroku je možné zvoliť aké komponenty budú v balíčku obsiahnuté. Tiež je potrebné výsledný balíček pomenovať. Pre pomenovanie je v ROS zavedená konvencia, že názov tohto balíčka by mal mať formát *"názov pôvodného balíčka"_moveit_config*. V našom prípade je názov nášho balíčka *melfa_rv-6sl_moveit_config*. Následne je možné vygenerovať výsledný balíček. Výsledný balíček je potrebné vložiť do zložky workspacu ROS, ktorá obsahuje aj zdrojové súbory.



Obr. 4.9: Záverečný krok vytvárania balíčka v MSA.

Demo príklad

Overiť funkčnosť celého balíčku môžeme pomocou demo príkladu. Pomocou tohto príkladu overíme správnu funkčnosť vizualizácie v RViz a plánovanie trasy robotického ramena. Pred spustením samotného príkladu je potrebné balíček preložiť a následne spustiť skript

potrebný pre nastavenie prostredia pomocou nasledujúcich príkazov z koreňového adresára použitého workspacu:

```
catkin_make
source devel/setup.bash
roslaunch melfa_rv-6sl_moveit_config demo.launch use_gui:=true
```

Po spustení vizualizačného prostredia RViz s pluginom MotionPlaning môžeme overiť funkčnosť plánovania, rozsahy pohybu jednotlivých kĺbov a detekciu kolízií. V tomto prípade pracuje vizualizácia s falošnou spätnou väzbou.

Pri spustení sme pomocou argumentu `use_gui:=true` spustili jednoduché grafické rozhranie pre uzol */joint_state_publisher*, ktorý publikuje nastavene hodnoty kĺbov do topicu */joint_states*. Pomocou tohto rozhrania je tak možné meniť pozíciu robotického ramena, nakoľko naša vizualizácia pracuje s falošnými spätnými väzbami.

```
jakub@Surface:~$ rostopic info /joint_states
Type: sensor_msgs/JointState

Publishers:
 * /melfa_driver (http://Surface:37337/)

Subscribers:
 * /robot_state_publisher (http://Surface:43003/)
 * /move_group (http://Surface:34967/)
```

Obr. 4.10: Výpis pripojených topicov k uzlu */joint_states* po spustení dema.

4.5 Spúšťač MoveIt!

Pre spustenie MoveIt! je potrebný spúšťací súbor rovnako ako v prípade ostatných uzlov v ROS. Obsah týchto súborov je zapísaný pomocou značkovacieho jazyka XML. Pre potreby práce sme vytvorili komplexný spúšťací súbor, ktorý okrem MoveIt! spustí tiež vizualizačné prostredie a na základe vstupných argumentov tiež simulačné prostredie Gazebo alebo rozhranie potrebné pre komunikáciu s reálnym ramenom. V tejto časti sa budeme zaoberať len častou tohto súboru, ktorá má za úlohu spustiť MoveIt. Kompletný súbor si priblížime v časti venovanej implementácií 6.

Pre zabezpečenie správnej funkčnosti MoveIt, je potrebné načítať konfiguračné súbory z disku a nahrať ich do parametrického servera ROS. Načítame tak SRDF, vlastnosti kinematiky a vlastnosti kĺbov. Následne je možné spustiť samotný uzol `move_group`.

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

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

Kapitola 5

Ovládač pre Mitsubishi Melfa

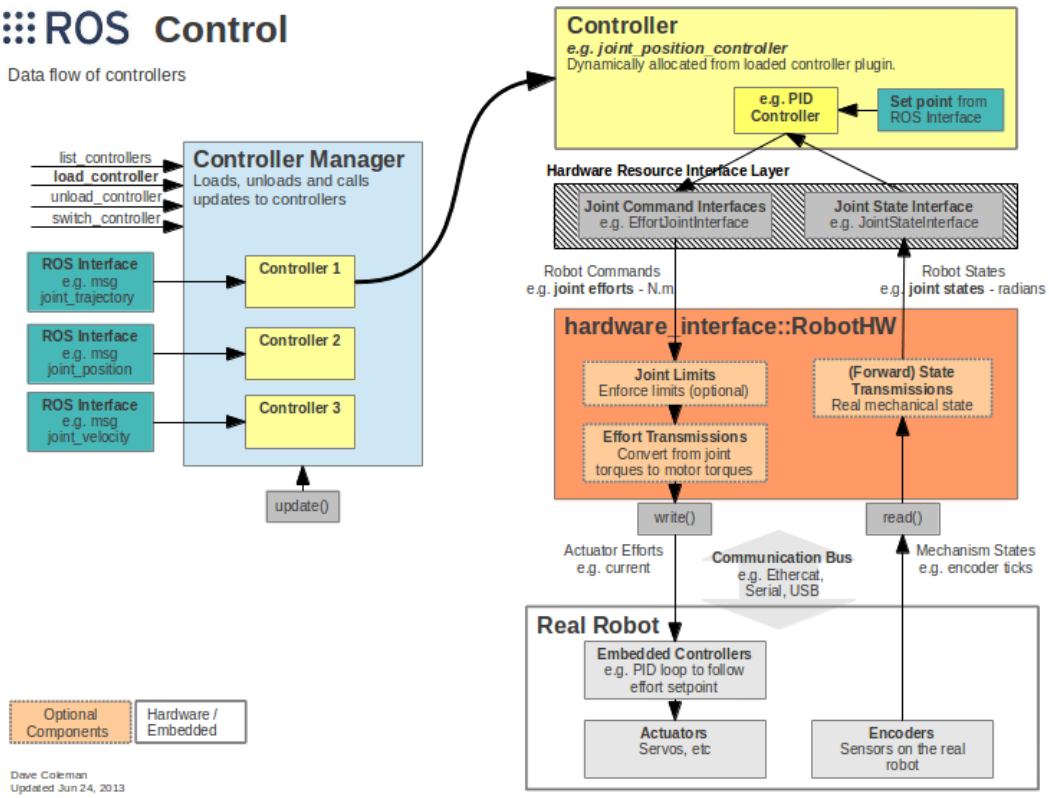
Pre komunikáciu s hardwarom využíva framework ROS hardwarové rozhrania, označované tiež hardware interfaces, pomocou ktorých je možné riadiť servomotory a riadiace jednotky niekoľkých výrobcov. Túto podporu [40] následne s pomocou ROS–Industrial rozširujeme aj na niektoré priemyselné ramená renomovaných výrobcov ako ABB, Adept, Fanuc, Motoman a kolaboratívne robotické ramená značky Universal Robots.

Robotické ramená Mitsubishi Melfa však touto priamou podporou nedisponujú. V tomto prípade však, vďaka komunite, potrebné rozhranie vzniklo. Je dostupné vo forme repozitára *melfa_robot*[45], konkrétne sa jedná o ovládač *melfa_driver*. Pomocou tohto ovládača je tak možné komunikovať s kontrolérom robotického ramena a riadiť jeho pohyb pomocou ROS a MoveIt!. Bližšie sa tomuto ovládaču budeme venovať v jednej nasledujúcich častí textu 5.1.1. Najskôr si však vysvetlíme základy riadenia hardwaru pomocou ROS Control [42].

5.1 ROS Control

Hardwarové rozhranie využívané v ROS je súčasťou balíčka *ros_control*. Súčasťou tohto balíčka je uzol *controller_manager*, ktorého úlohou je správa programových kontrolérov, ktoré sme si spomínali už v prípade MSA 4.4.1, keď sme počas konfigurácie nastavovali práve kontroléry ROS Control. Tieto kontroléry následne komunikujú s MoveIt!, RViz, užívateľskými a hardwarovými rozhraniami.

ROS Control obsahuje niekoľko typov kontrolérov a hardwarových rozhraní. Dokážeme tak pokryť celú škálu fyzických vlastností hardwaru, ktorý chceme ovládať. Je tak možné riadiť rýchlosť pohybu, použitú silu, pozíciu a trajektóriu robotov a robotických ramien. Nesmieme však zabúdať, že pre potreby riadenia hocijakého hardwaru je tiež nutné poznať aj stav, v akom sa tento hardware nachádza. Pomocou rozhraní a kontrolérov tak prebieha obojsmerná komunikácia medzi hardwarom a softwarom.



Obr. 5.1: Architektúra a tok dát v ROS Control, prevzaté z [42].

5.1.1 Hardwarové rozhranie

Kontroléry pre riadenie robotického ramena sme už vyriešili počas konfigurácie v MSA 4.4.1, ostáva už len doplniť hardwarové rozhranie. To v našom prípade zabezpečíme pomocou balíčka *melfa_driver*. Toto rozhranie sa rovnako ako ostatné balíčky spúšťa pomocou spúšťacieho súboru a následne beží v podobe ROS uzla.

Balíček `melfa_driver`

Balíček *melfa_driver* obsahuje niekoľko zdrojových súborov organizovaných v nasledujúcej štruktúre:

```
melfa_driver
├── config
│   └── controller.yaml
├── include
│   ├── melfa_hardware_interface.h
│   └── strdef.h
├── launch
│   └── melfa_driver.launch
├── src
│   ├── melfa_driver_node.cpp
│   ├── melfa_hardware_interface.cpp
│   └── melfa_loopback_node.cpp
├── test
│   ├── joint_trajectory_controller.test
│   └── test_joint_trajectory_controller.py
├── CHANGELOG.rst
├── CMakeLists.txt
├── export.log
└── package.xml
```

Pred ich použitím je vhodné aspoň v skratke popísať funkcionalitu, ktorú jednotlivé súbory pokrývajú:

- **controller.yaml** - konfiguračný súbor, obsahujúci kontroléry určené pre využitie s ramenamami melfa, ktoré používal tvorca repozitára
- **melfa_hardware_interface.h** - hlavičkový súbor obsahujúci potrebné deklarácie pre preklad melfa_hardware_interface.cpp
- **strdef.h** - hlavičkový súbor, obsahujúci deklarácie dátových štruktúr využívaných v komunikácii s kontrolérom Melfa
- **melfa_driver.launch** - spúšťací súbor ovládača
- **melfa_driver_node.cpp** - zdrojový súbor ROS uzlu, implementujúci ovládač pre roboty Melfa
- **melfa_hardware_interface.cpp** -zdrojový súbor obsahujúci definície tried typu `MelfaHW`
- **melfa_loopback_node.cpp** - zdrojový súbor ROS uzlu, ktorý vytvára falošnú spätnú väzbu pre potreby testovania
- **joint_trajectory_controller.test** - spúšťač testovacieho skriptu
- **test_joint_trajectory_controller.py** - testovací skript v jazyku Python

Uzol melfa_driver

Ovládač *melfa_driver* je program, ktorý reprezentuje uzol v distribuovanom meta-systéme ROS. Po spustení tohto programu pomocou spúšťacieho súboru, je toto jeho chovanie následovné:

1. program zaregistrovaný do siete ROS ako uzol.
2. následne sú zaregistrované potrebné parametre v parametrickom serveri ROS.
3. po splnení prechádzajúcich krokov je vytvorené hardwarové rozhranie typu `MelfaHW` a `controller_manager`.
4. po vytvorení rozhrania je možné vytvoriť objekt, ktorý zabezpečí potrebné diagnostické funkcie uzlu.
5. ak sme spustili uzol s aktivovanou realtime funkcionalitou 5.2, v tomto kroku dochádza k overeniu dostupnosti potrebných systémových požiadaviek a nastaveniu potrebných parametrov.
6. nastavenie parametrov, inicializácia a spustenie sieťového rozhrania
7. zaslanie prvej správy neobsahujúcej pozičné dáta, zahajujúcej komunikáciu 5.3 s kontrolérom robotického ramena
8. periodicky sa opakujúca slučka, ktorá príjme dáta od kontroléru robotického ramena, následne predá tieto dáta hardwarovému rozhraniu. Po ukončení prijímania dát následne odošle nové pozičné dáta kontroléru robotického ramena. Táto slučka je nekonečná a ukončiť beh tohto uzlu je možné len zaslaním signálu prerušenia.

5.2 Real time

Real time, alebo tiež reálny čas, je podstatné meno, ktoré označuje čas, počas ktorého niečo prebieha, nadobúda určitý stav [22]. Ak tento termín použijeme v súvislosti s počítačovým systémom, aplikáciou alebo procesom, používame tento výraz vo forme prídavného mena a popisujeme tak spôsob reakcie na vstupne požiadavky. Reakcia, ktorú očakávame, by mala mať malé oneskorenie odpovede na požiadavku. Najpresnejšie je možné tento termín definovať pomocou real time systému [53]. Real time systém, je taký systém, ktorý garantuje odpovede na požiadavky s minimálnym oneskorením a s ohraničením maximálneho času na odpoveď, tiež nazývaného deadline. Takýto systém musí tiež produkovať funkčné správne výstupné odpovede na vstupné požiadavky.

Cieľom tejto bakalárskej práce je real time riadenie robotického ramena za pomoci ROS. Operačný systém Linux, ktorý využívame pre beh meta-operačného systému ROS, za bežných okolností nenadobúda vlastnosti realtime systému, z dôvodu nastavenia plánovacej politiky pre prideľovanie výpočtového času na procesore pre bežiace procesy. Pre docielenie real time správania systému je preto potrebné zmeniť plánovaciu politiku. Preto je potrebné aplikovať real time patch kernelu. Po aplikovaní tohto patch, označovaného tiež RT-Preempt, zmeníme chovanie jadra na plne predvídateľné. Ako aplikovať takýto patch je popísane v článkoch [16] a [30].

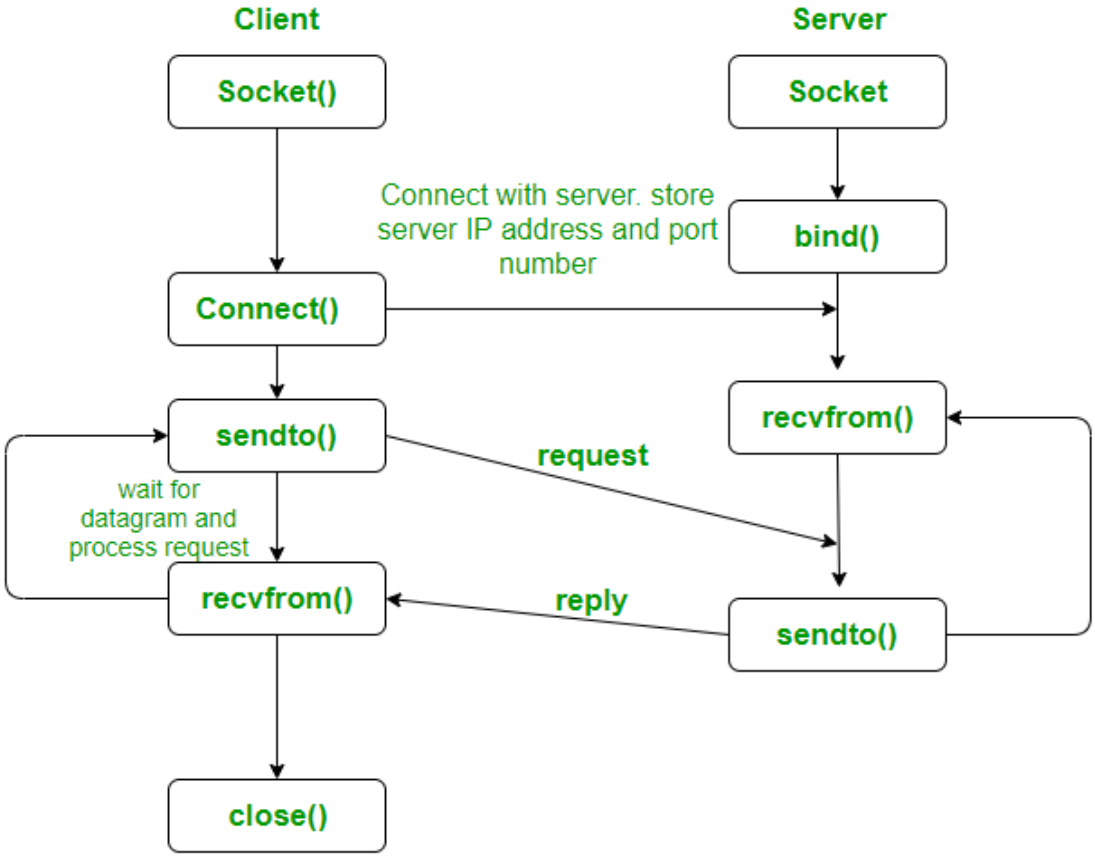
Pre nastavenie tejto novej politiky plánovača pre beh programu, je potrebné zavolať v zdrojovom kóde funkciu `sched_setscheduler()` s príslušnými parametrami.

5.3 Sieťová komunikácia

Komunikácia s robotickým ramenom, respektíve kontrolórom tohto ramena a počítačom prebieha pomocou lokálnej siete Ethernet. Bez tohto spojenia by nebolo možné zabezpečiť výmenu správ obsahujúcich pozičné dáta medzi kontrolórom a počítačom. Vzájomná komunikácia medzi kontrolérom a počítačom prebieha pomocou protokolu UDP. Protokol UDP vyžaduje pre svoju činnosť dva prvky, klienta a server. V našom prípade kontolér robotického ramena (Mitsubishi CR2B-574) zastáva úlohu servera a počítač úlohu klienta.

Protokol UDP má jednoduchú schému činnosti, viď obrázok 5.2. Klient zašle pomocou funkcie `sendto()` UDP datagram na server a následne volá funkciu `recvfrom()`, ktorá čaká na príchodzí datagram s odpoveďou. Pri odosielaní je nutné špecifikovať adresu príjemcu, na ktorú sa datagramy zasielajú. Súčasťou datagramu klienta je aj jeho vlastná IP adresa, aby server vedel, komu je potrebné zaslať odpoveď. Ak klient pri prvotnom nadviazaní komunikácie nešpecifikoval lokálny port (`bind()`), jadro systému mu pridelí automaticky jeden z aktuálne voľných.

Pri komunikácii počítača s kontrolérom port nešpecifikujeme. Vyžaduje sa len IP adresa zariadení. V prípade kontroléru je jeho IPv4 adresa 192.168.0.1. Pre nadviazanie komunikácie je potrebné, aby mal počítač nastavenú adresu v rovnakej sieti, napríklad 192.168.0.2. Po nastavení je možné funkčnosť spojenia overiť pomocou nástroja pinč z príkazového riadku.



Obr. 5.2: UDP protokol v C, prevzaté z [25]

Kapitola 6

Implementácia

Implementácia funkčného riešenia pozostáva z niekoľkých krokov, ktoré vedú k funkčnému riešeniu pre riadenie robotického ramena Mitsubishi Melfa za pomoci ROS. Veľká časť týchto krokov je viac či menej spomenutá v predchádzajúcich častiach textu, preto sa na nej budeme v tejto časti odkazovať.

6.1 Postup pri implementácii

Prvým krokom je vytvorenie kvalitného modelu robotického ramena vo formáte URDF 2.1.5. Tento model je kľúčový, pretože bez neho by nebolo možné použiť väčšinu nástrojov, ktoré sú potrebné pre plánovanie trajektórie alebo riadenie ramena, nezáleží na tom či simulovaného, alebo reálneho. Preto je nutné venovať dostatočné množstvo času a pozornosti príprave 3D modelu v CAD softwari a zhromaždeniu potrebných informácií o parametroch robotického ramena, ktoré chceme riadiť. Náš 3D model sme postupne upravili do takej podoby, že nesie v sebe maximum možných informácií o reálnom ramene, ktoré je možné do 3D modelu zaniesť v rámci CAD softwaru. Následne sme z tohto 3D modelu vygenerovali detailný popis ramena pomocou pluginu SolidWorks to URDF a doplnili zvyšok potrebných informácií o reálnom robotickom ramene.

Ďalším krokom je vytvorenie konfiguračného balíčka pre plánovač trajektórie MoveIt! 4. Pre tieto účely existuje samostatný nástroj MoveIt! setup assistant, ktorý intuitívne prevedie užívateľa takmer celým procesom. Potrebné informácie k jednotlivým krokom je možné nájsť v kapitole, venovanej tomuto plánovaču 4. Po vytvorení balíčka máme už čiastočne funkčné riešenie, ktoré dokáže v tomto kroku plánovať trajektóriu a ovládať simulované robotické rameno.

Ďalším krokom je vytvorenie rozhrania, ktoré nám zabezpečí komunikáciu s reálnym ramenom, vďaka čomu ho bude možné ovládať za pomoci ROS. V našom prípade sme prevzali tuto časť z repozitára `melfa_robot` a zakomponovali sme ju do nášho riešenia. Jedná sa konkrétne o balíček s názvom `melfa_driver` 5.1.1. Tiež sme v tomto kroku zaistili real time funkcionalitu pomocou patchu kernelu systému Ubuntu.

Doteraz sme jednotlivé balíčky v jednotlivých krokoch spúšťali pomocou samostatných spúšťacích súborov. Pre zjednodušenie použitia sme tak vytvorili jeden komplexný spúšťací súbor pomenovaný `melfa_rv-6sl_execution.launch`, ktorý na základe zvolených argumentov spúšťa potrebné funkcie. Taktiež sme vytvorili novú konfiguráciu užívateľského rozhrania vizualizačného nástroja RViz, aby pri spustení automaticky načítal všetky potrebné pluginy a nástroje.

Posledným krokom implementácie bolo vytvorenie rozšírenej možnosti plánovania trasy robota, pomocou vytvorenia jednoduchého rozhrania, ktoré umožňuje zadávať cieľové polohy robotického ramena aj pomocou súradníc v priestore.

6.2 Spúšťač výsledného riešenia

Spúšťač výsledného riešenia bol vytvorený prevzatím potrebných častí spúšťacích súborov, ktoré slúžia k spúšťaniu jednotlivých nástrojov samostatne. Vo výsledku je tak možné spustiť pomocou tohto súboru buď simulátor Gazebo alebo rozhranie pre ovládanie reálneho robotického ramena `melfa_driver`. Tiež je možné spustiť `loopback_node`, ak chceme pracovať s rozhraním pre robotické rameno, ale rameno nemáme pripojené.

Tento spúšťač tiež zabezpečuje nahratie potrebných parametrov z disku na parametrický server ROS, ako aj spustenie potrebných kontrolérov ROS Control a plánovača MoveIt!.

Na záver nesmieme zabudnúť na spustenie vizualizačného prostredia RViz, ktoré nám umožňuje ovládať funkcie plánovača a zobrazuje nám aktuálny stav robota.

Kapitola 7

Testovanie

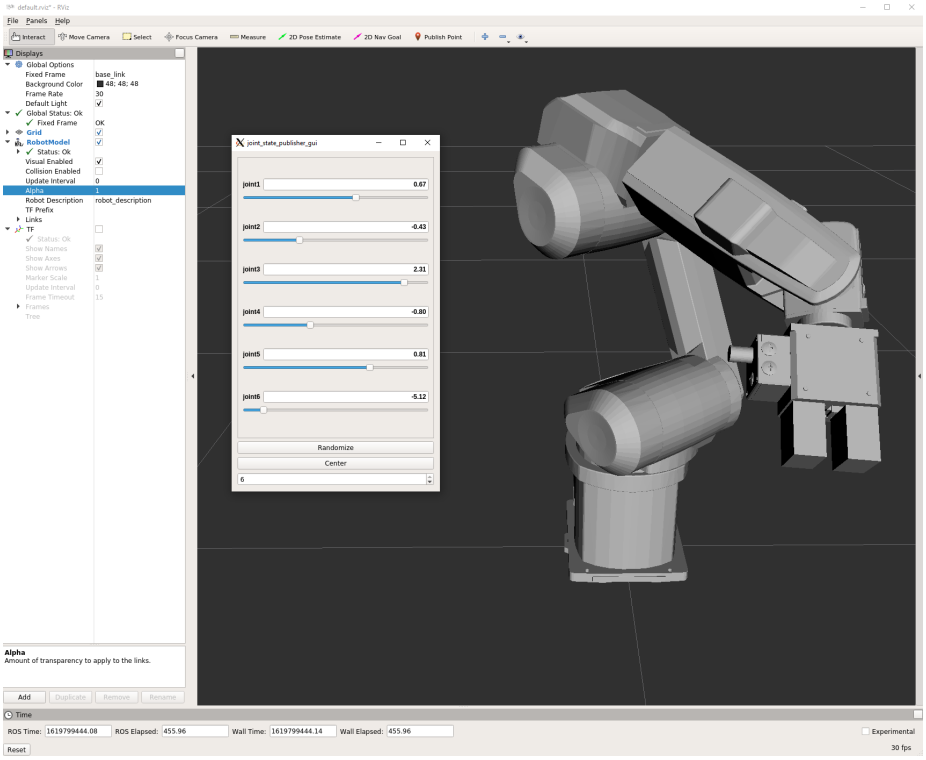
Testovanie prebiehalo v niekoľkých fázach a po každom ďalšom kroku v implementácii. Prvotné fázy testovania prebiehali len v simulačnom prostredí. Až neskôr s pridaním hardwarového rozhrania mohli prebehnúť aj testy s reálnym ramenom.

7.1 Testovanie URDF modelu - simulácia

Modelu robotického ramena bola venovaná veľká pozornosť pri jeho vytváraní a testovaní. Odpovedá tomu aj rozsah kapitoly venovanej robotickému ramenu a modelom. Pri prvom teste sa nám okamžite po spustení vizualizačného prostredia RViz naskytol nepríjemný pohľad. Po vložení robotického ramena do simulačného okna sa namiesto robotického ramena zobrazila len hromada segmentov. Riešenie bolo vcelku jednoduché. Bolo potrebné nastaviť, ktorý segment ramena je základňa. Následne začalo testovanie rozsahu pohybu kĺbov, pomocou rozhrania *joint_state_publisher-GUI*, viď obrázok 7.1, ktoré sa automaticky zobrazí pri spustení vizualizačného prostredia pomocou spúšťacieho súboru `display.launch`. Po otestovaní rozsahov a správnosti smeru pohybov jednotlivých kĺbov sme mali prvú fázu ukončenú, nakoľko sme nespozorovali žiadne chyby.

7.2 Testovanie MoveIt! - simulácia

Keďže sme pri prvej fáze testovania neobjavili žiadne chyby, pokračovali sme v implementácii na ďalší krok. Vygenerovali sme balíček potrebný pre plánovač MoveIt! Pri testovaní sme narazili na zvláštne chovanie tretieho kĺbu. Umožňoval natočenie kĺbu do polohy, do ktorej by sa pri ramene nedokázal natočiť, pretože by ho zastavil mechanicky doraz. Preto sme pri najbližšej príležitosti navštívili miestnosť s reálnym robotickým ramenom na fakulte. Ukázalo sa, že model ramena má nulový bod pootočený o 90° v kladnom smere otáčania voči reálnemu ramenu. Po opätovnom otestovaní modelu podľa prvej fázy testovania, sa tento problém prejavil, čo sme pôvodne prehliadli. Po hlbšom pátraní a študovaní dokumentácie sme si potvrdili, že náš nulový bod je v modeli naozaj zrotovaný o 90° v kladnom smere otáčania. K tomuto omylu došlo na základe toho, že robotické rameno nemá na všetkých kĺboch kalibračné rysky v nulových polohách kĺbov. Tretí kĺb sa kalibruje v pozícii +90°. Taktiež k tomuto omylu prispel fakt, že rameno je na všetkých obrázkoch v dokumentácii zobrazované v kalibračnej polohe, nie nulovej, o čom nie je v texte manuálu žiadna zmienka. Tento fakt je možné odčítať len z nákresu maximálnych rozsahov pohybu.



Obr. 7.1: Vizualizačné prostredie RViz a interface joint_state_publisher-GUI

Táto chyba znamenala vrátenie sa na úplný začiatok a prerobenie modelu v Solidwork, následne bolo nutné zopakovať všetky kroky, až k vygenerovaniu a testovaniu konfigurácie pre plánovač MoveIt!. Tentokrát už všetko fungovalo správne. Testy prebehli aj za pomoci simulátora Gazebo. Bolo tak možné pokračovať v implementácii na vytvorenie rozhrania pre komunikáciu.

7.3 Testovanie melfa_driver

Po úspešnom integrovaní ovládača melfa_drive do našej implementácie, sme otestovali tento ovládač najskôr v domácich podmienkach s využitím loopback_node. Následne sme pristúpili k testovaniu tohto rozhrania na reálnom robotickom ramene. Pri prvom teste dochádzalo k výpadkom komunikácie medzi kontrolérom CR2B-574 a systémom ROS. Problém spôsoboval už spomínaný plugin pre simulátor Gazebo umiestnený v URDF súbore. Ten spôsoboval nestabilitu programových kontrolérov, čo viedlo k výpadkom. Vytvorenie URDF súboru bez tohto pluginu pre potreby ovládania reálneho ramena vyriešilo tento problém. Pri opätovnom testovaní už nedošlo k žiadnemu výpadku a bolo možné otestovať rôzne scenáre plánovania trajektórie robotického ramena. Všetky testy prebiehali len pomocou vizuálneho prostredia RViz. Tieto testy prebehli úspešne, no vyplynula potreba vytvorenia rozhrania, ktoré umožní zadávanie cieľových polôh na základe zadaných súradníc v priestore.

7.4 Testovanie rozhrania melfa_interface

Poslednou fázou testovania nebolo otestovanie ovládača melfa_driver, ale testovanie plánovania trasy pomocou zadaných súradníc. V domácich podmienkach v simulačnom prostredí všetko fungovalo správne. Bolo však potrebné otestovať plánovanie aj v reálnych podmienkach. Najmä bolo potrebné otestovať prípad, ak sa nástroj ramena pohybuje v tesnej blízkosti zeme, nakoľko sme uchopovacie prsty modelovali len ako boundryboxy. Hrozila tak reálna šanca, že mohlo dôjsť k chybe merania, čo by spôsobilo náraz do podložky. Tento test však prebehol úspešne. Boundryboxy sú namodelované z rezervou 2mm (sú väčšie), vďaka čomu antikolízny systém nedovolí robotovi naraziť do podložky. V simulácii mu povolí len dotyk, ak je nastroj paralelne s podložkou.

Kapitola 8

Záver

Cieľom tejto bakalárskej práce bolo vytvorenie funkčnej implementácie riadenia robotického ramena Mitsubishi Melfa RV-6SL za pomoci ROS s využitím plánovača MoveIt!. Pre zaistenie požadovanej funkčnosti tak bolo potrebné zakomponovať do implementácie veľké spektrum nástrojov, funkcií a možností ROS. Stretli sme sa tak v tejto bakalárskej práci s nutnosťou úpravi 3D modelov v CAD softwaroch a vytváraním popisov robotického ramena v rôznych textových formátoch. Tiež bolo potrebné vytvárať a upravovať konfiguračné súbory a balíčky. Nesmieme tiež zabudnúť na testovanie, pri ktorom sa odhaľovali chyby, ktoré nie vždy mali na prvý pohlaď očividné riešenie. Nakoniec však naša práca nevyšla na zmar a podarilo sa nám vytvoriť funkčné riešenie.

Výsledná implementácia zvláda plánovanie trajektórie pomocou plánovača MoveIt! a následne túto trajektóriu dokáže prakticky vykonať na reálnom ramene pomocou rozhrania melfa_driver.

Výhodou pri práci na implementácii bola tiež skutočnosť, že vizualizačné prostredie RViz a simulátor Gazebo dokážu naozaj kvalitne simulovať robotické platformy. Avšak je nutné podotknúť, že kvalita simulácie je závislá aj na jej príprave, ktorej sme venovali dostatok času. Kvôli pandémii COVID-19 bol sťažený prístup na fakultu k reálnemu ramenu, ale vďaka simulátorom to nebola až taká obrovská prekážka. Toto je nesporná výhoda, ktorá umožňuje zjednodušiť vývoj robotických aplikácií založených na ROS v budúcnosti.

Taktiež nesmieme zabúdať aj na ďalšie možnosti tohto frameworku, ako je podpora kolaboratívnej robotiky, strojového videnia a ďalších nových technológií, ktoré prenikajú do sveta robotiky a ROS im vychádza v ústrety. Nakoľko sa naša fakulta zaoberá do veľkej miery týmito oblasťami, má táto bakalárska práca vysoký potenciál stať sa základným kameňom pre budúce práce zaoberajúce sa strojovým videním s využitím ROS.

Táto bakalárska práca mala zaujímavý charakter vďaka tomu, že sa zaoberala open-source projektom, ktorého cieľom je preniknúť na trh priemyselnej automatizácie a otvoriť tak tento uzavretý svet širšej odbornej verejnosti alebo aj hobby nadšencom.

Literatúra

[1] *URDF transmission for simulation and hardware_interface - ROS Answers: Open Source Q&A Forum*. [cit. 2021-05-08]. Dostupné z: https://answers.ros.org/question/346945/urdf-transmission-for-simulation-and-hardware_interface/.

[2] *Melfa Industrial Robots Instruction Manual*. Revision J. Tokyo building, 2-7-3, Marunouchi, Chiyoda-Ku, Tokyo 100-8310, Japan: Mitsubishi Electric Corporation, 2003.

[3] *Mitsubishi Industrial Robot RV-6S Series Standard Specifications Manual*. Revision D. Tokyo building, 2-7-3, Marunouchi, Chiyoda-Ku, Tokyo 100-8310, Japan: Mitsubishi Electric Corporation, 2006.

[4] *Mitsubishi Industrial Robot RV-6S Series Instruction manual Robot arm setup & maintance*. Revision C. Tokyo building, 2-7-3, Marunouchi, Chiyoda-Ku, Tokyo 100-8310, Japan: Mitsubishi Electric Corporation, 2009.

[5] *CR750/CR751 Series Controller Instruction Manual*. Revision E. Tokyo building, 2-7-3, Marunouchi, Chiyoda-Ku, Tokyo 100-8310, Japan: Mitsubishi Electric Corporation, 2013.

[6] Belica, A. *Možnosti měření zatěžovacího momentu u elektrických strojů*. Vysoké učení technické v Brně. Fakulta elektrotechniky a komunikačních technologií, 2014 [cit. 2021-05-04].

[7] Božiková, M., Kubík Ľubomír, Hlaváčová, Z., Csillag, J., Petrovic, A. et al. *Multimediálna učebnica fyziky pre technikov 2. diel* [Slovenská poľnohospodárska univerzita v Nitre, 2019. [Online].]. [cit. 2021-05-04]. ISBN 978-80-552-2118-2. Dostupné z: http://dl.slpk.sk/fyzika2.

[8] DailyAutomation. *04 Základy Priemyselnej robotiky- Základné delenie robotov*. Jan 2020 [cit. 2021-05-03]. Dostupné z: https://www.dailyautomation.sk/01-zaklady-priemyselnej-robotiky-zakladne-delenie/.

[9] Dassault Systèmes SolidWorks Corporation. *SOLIDWORKS*. 2021. Dostupné z: https://www.solidworks.com/.

[10] Gazebo. *Gazebo : Tutorial : ROS control*. Open Source Robotics Foundation [cit. 2021-03-26]. Dostupné z: http://gazebosim.org/tutorials/?tut=ros_control.

[11] Gazebo. *Why Gazebo?* [cit. 2021-05-03]. Dostupné z: http://gazebosim.org/.

[12] Gonzalez, C. *What's the Difference Between Industrial Robots?* Dec 2016 [cit. 2021-05-03]. Dostupné z: https://www.machinedesign.com/markets/robotics/article/21835000/whats-the-difference-between-industrial-robots.

[13] International Organization for Standardization. *ISO 10218-1:2011(en) Robots and robotic devices — Safety requirements for industrial robots — Part 1: Robots.* International Organization for Standardization, Jul 2011 [cit. 2021-05-03]. Dostupné z: https://www.iso.org/obp/ui/#iso:std:iso:ts:15066:ed-1:v1:en.

[14] International Organization for Standardization. *ISO 10218-2:2011(en) Robots and robotic devices — Safety requirements for industrial robots — Part 2: Robot systems and integration.* International Organization for Standardization, Jul 2011 [cit. 2021-05-03]. Dostupné z: https://www.iso.org/obp/ui/#iso:std:iso:10218:-2:ed-1:v1:en.

[15] International Organization for Standardization. *ISO/TS 15066:2016(en) Robots and robotic devices — Collaborative robots.* International Organization for Standardization, Feb 2016 [cit. 2021-05-03]. Dostupné z: https://www.iso.org/obp/ui/#iso:std:iso:ts:15066:ed-1:v1:en.

[16] Kautilya, C. *How to setup PREEMPT RT on Ubuntu 18.04 | Chenna Kautilya /home/hashb.* chenna.me, Nov 2020 [cit. 2021-04-18]. Dostupné z: https://chenna.me/blog/2020/02/23/how-to-setup-preempt-rt-on-ubuntu-18-04/.

[17] Kazda, L. *Měření momentu setrvačnosti.* České vysoké učení technické. Fakulta strojní, 2015 [cit. 2021-05-04].

[18] Kumar, A. *An Introduction to Commercial Robotic Control with ROS Industrial - Technical Articles.* Control Automation, Jul 2018 [cit. 2021-03-03]. Dostupné z: https://control.com/technical-articles/an-introduction-to-ros-industrial-for-the-commercial-use-of-ros/.

[19] Lentin, J. *ROS Robotics Projects* [Packt, 2017. [Online].]. [cit. 2021-05-04]. ISBN 9781783554713. Dostupné z: https://www.packtpub.com/product/ros-robotics-projects/9781783554713.

[20] Lu, X. *Scheme of the Delta robot.* May 2016 [cit. 2021-05-03]. Dostupné z: https://www.researchgate.net/figure/Scheme-of-the-Delta-robot_fig3_303469087.

[21] Manic, M. *TRIGLIDE parallel robot with 3 DOF.* Jun 2008 [cit. 2021-05-03]. Dostupné z: https://www.researchgate.net/figure/TRIGLIDE-parallel-robot-with-3-DOF_fig1_4356741.

[22] Merriam-Webster. *Real Time | Definition of Real Time by Merriam-Webster.* Merriam-Webster [cit. 2021-05-03]. Dostupné z: https://www.merriam-webster.com/dictionary/real%20time.

[23] Mitsubishi Electric Corporation. *Mitsubishi Electric Europe e-shop.* 2021. Dostupné z: https://mitsubishi-electric-eshop.mee.com//.

[24] Mitsubishi Electric Corporation. *Mitsubishi Electric Factory Automation - Czech Republic.* 2021. Dostupné z: https://cz3a.mitsubishielectric.com/fa/cs/.

[25] Mohityadav. *UDP Client Server using connect | C implementation*. GeekforGeeks, Apr 2017 [cit. 2021-05-05]. Dostupné z: https://www.geeksforgeeks.org/udp-client-server-using-connect-c-implementation/.

[26] MoveIt. *Concepts*. PickNik Robotics [cit. 2021-05-06]. Dostupné z: https://moveit.ros.org/documentation/concepts/.

[27] MoveIt. *Moving robots into the future*. [cit. 2021-05-03]. Dostupné z: https://moveit.ros.org/.

[28] ROS. *MoveIt! Setup Assistant Tutorial — moveit_setup_assistant_tutorial documentation*. ROS [cit. 2021-04-02]. Dostupné z: http://docs.ros.org/en/hydro/api/moveit_setup_assistant/html/doc/tutorial.html#.

[29] ROS. *Powering the world's robots*. [cit. 2021-05-03]. Dostupné z: https://www.ros.org/.

[30] ROS Wiki contributors. *Building realtime Linux for ROS 2 [community-contributed] — ROS 2 Documentation: Foxy documentation*. Open Robotics [cit. 2021-04-18]. Dostupné z: https://docs.ros.org/en/foxy/Tutorials/Building-Realtime-rt_preempt-kernel-for-ROS-2.html.

[31] ROS Wiki contributors. *Sw_urdf_exporter/Tutorials/Export an Assembly*. Open Robotics [cit. 2021-04-12]. Dostupné z: http://wiki.ros.org/sw_urdf_exporter/Tutorials/Export%20an%20Assembly.

[32] ROS Wiki contributors. *ROS/Concepts*. Open Robotics, Apr 2014 [cit. 2021-05-03]. Dostupné z: http://wiki.ros.org/ROS/Concepts.

[33] ROS Wiki contributors. *Urdf/XML/model*. Open Robotics, Dec 2016 [cit. 2021-04-16]. Dostupné z: http://wiki.ros.org/urdf/XML/model.

[34] ROS Wiki contributors. *Kinetic*. Open Robotics, Jan 2018 [cit. 2021-05-08]. Dostupné z: http://wiki.ros.org/kinetic.

[35] ROS Wiki contributors. *Melodic*. Open Robotics, Aug 2018 [cit. 2021-05-08]. Dostupné z: http://wiki.ros.org/melodic.

[36] ROS Wiki contributors. *Rviz*. Open Robotics, May 2018 [cit. 2021-03-25]. Dostupné z: http://wiki.ros.org/rviz.

[37] ROS Wiki contributors. *Urdf/XML/joint*. Open Robotics, Nov 2018 [cit. 2021-04-16]. Dostupné z: http://wiki.ros.org/urdf/XML/joint.

[38] ROS Wiki contributors. *Urdf*. Open Robotics, Jan 2019 [cit. 2021-03-09]. Dostupné z: http://wiki.ros.org/urdf.

[39] ROS Wiki contributors. *Industrial*. Open Robotics, May 2020. Dostupné z: http://wiki.ros.org/Industrial.

[40] ROS Wiki contributors. *Industrial/supported_hardware*. Open Robotics, Oct 2020 [cit. 2021-03-25]. Dostupné z: http://wiki.ros.org/Industrial/supported_hardware.

[41] ROS Wiki contributors. *Noetic*. Open Robotics, Aug 2020 [cit. 2021-05-08]. Dostupné z: http://wiki.ros.org/noetic.

[42] ROS Wiki contributors. *Ros_control*. Open Robotics, Aug 2020 [cit. 2021-04-08]. Dostupné z: http://wiki.ros.org/ros_control.

[43] SCHUNK GmbH & Co. KG. *PG 70*. Apr 2021 [cit. 2021-05-04]. Dostupné z: https://schunk.com/dk_en/gripping-systems/product/2493-0306095-pg-70/.

[44] Tajima, R. a Vollprecht, W. *Melfa Driver*. Feb 2019 [cit. 2021-05-03]. Dostupné z: https://github.com/tork-a/melfa_robot/tree/master/melfa_driver.

[45] Tajima, R., y.suzuki, Hiraizumi, K. a Vollprecht, W. *Melfa Robot*. Feb 2019 [cit. 2021-05-03]. Dostupné z: https://github.com/tork-a/melfa_robot.

[46] Wikipedia contributors. *XML*. Wikimedia Foundation, May 2017. Dostupné z: https://sk.wikipedia.org/wiki/XML.

[47] Wikipedia contributors. *R.U.R.* Wikimedia Foundation, Nov 2019 [cit. 2021-05-03]. Dostupné z: https://sk.wikipedia.org/wiki/R.U.R.

[48] Wikipedia contributors. *Denavit–Hartenberg parameters*. Wikimedia Foundation, Oct 2020 [cit. 2021-04-20]. Dostupné z: https://en.wikipedia.org/wiki/Denavit%E2%80%93Hartenberg_parameters.

[49] Wikipedia contributors. *Krútiaci moment*. Wikimedia Foundation, Dec 2020 [cit. 2021-05-04]. Dostupné z: https://sk.wikipedia.org/wiki/Kr%C3%BAtiaci_moment.

[50] Wikipedia contributors. *BASIC*. Wikimedia Foundation, May 2021 [cit. 2021-05-03]. Dostupné z: https://sk.wikipedia.org/wiki/BASIC.

[51] Wikipedia contributors. *ISO 10303-21*. Wikimedia Foundation, Feb 2021 [cit. 2021-04-23]. Dostupné z: https://en.wikipedia.org/wiki/ISO_10303-21.

[52] Wikipedia contributors. *Karel Čapek*. Wikimedia Foundation, Feb 2021 [cit. 2021-05-03]. Dostupné z: https://sk.wikipedia.org/wiki/Karel_%C4%8Capek.

[53] Wikipedia contributors. *Real-time computing*. Wikimedia Foundation, Apr 2021 [cit. 2021-05-03]. Dostupné z: https://en.wikipedia.org/wiki/Real-time_computing.

[54] Wikipedia contributors. *Robot*. Wikimedia Foundation, Apr 2021 [cit. 2021-05-03]. Dostupné z: https://sk.wikipedia.org/wiki/Robot.

[55] Wikipedia contributors. *STL*. Wikimedia Foundation, May 2021 [cit. 2021-05-03]. Dostupné z: https://sk.wikipedia.org/wiki/STL.

[56] Wágner, P. *Metoda plánování trajektorie robota v reálném čase*. VŠB-TU Ostrava, Fakulta elektrotechniky a informatiky, 2014 [cit. 2021-05-04].

[57] Yeshmukhametov, A., Kalimoldayev, M., Mamyrbayev, O. a Amirgaliev, Y. *Design and kinematics of serial/parallel hybrid robot | Semantic Scholar*. Semantic Scholar, 2017 [cit. 2021-03-14]. Dostupné z: https://www.semanticscholar.org/paper/Design-and-kinematics-of-serial%2Fparallel-hybrid-Yeshmukhametov-Kalimoldayev/84c59267845cb6477d314c20c77086ffe2d2925e.

Príloha A

Použitie

V tejto prílohe sú zhrnuté základné postupy, ktoré treba dodržať pre prípravu a spustenie implementácie.

Spustenie RViz a simulátora Gazebo

V koreňovom adresári workspacu:

```
catkin_make
source devel/setup.bash
roslaunch melfa_rv-6sl_moveit_config melfa_rv-6sl_execution.launch
sim:=true
```

Spustenie RViz s rozhraním melfa_driver

V koreňovom adresári workspacu:

```
catkin_make
source devel/setup.bash
roslaunch melfa_rv-6sl_moveit_config melfa_rv-6sl_execution.launch
```

Potrebné atribúty, ktoré je potrebné nastaviť na `true`:

- robot_ip - potrebné nastaviť adresu kontroléra robota, ak pracujeme s reálnym ramenom
- real_time - možnosť zapnúť real time správanie systémového plánovača
- loop_back - spustenie uzla pre spätnú väzbu, v prípade ak nepracujeme s robotickým ramenom

Spustenie rozhrania melfa_interface

Spustenie rozhrania pre zadávanie koncových polôh pomocou súradníc je potrebné, aby bola spustená simulácia alebo rozhranie pre reálneho robota. Následne je možné v ďalšom terminálovom okne spustiť toto rozhranie.

```
catkin_make
source devel/setup.bash
roslaunch melfa_interface melfa_interface.launch
```

Následne sa spustí rozhranie, pomocou ktorého je možné definovať koncové polohy.

RQT

RQT je nástroj, ktorý ponúka grafické rozhranie pre ovládanie kĺbov reálneho ramena. Je však potrebné, aby bol spúšťaný v separátnom terminálovom okne. Taktiež musí byť spustený ROS Master.

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