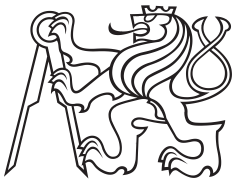
Bakalářská práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra kybernetiky

# Řídicí rozhraní pro roboty MITSUBISHI v prostředí ROS

Tomáš Chaloupecký

Školitel: Ing. Pavel Krsek, Ph.D.
Obor: Kybernetika a robotika
Květen 2021

# ZADÁNÍ BAKALÁŘSKÉ PRÁCE

## I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Chaloupecký** Jméno: **Tomáš** Osobní číslo: **483548**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra kybernetiky**

Studijní program: **Kybernetika a robotika**

## II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

**Řídicí rozhraní pro roboty MITSUBISHI v prostředí ROS**

Název bakalářské práce anglicky:

**Control Interface for MITSUBISHI Robots in ROS**

Pokyny pro vypracování:

1. Prostudujte způsob řízení manipulátorů a jejich rozhraní v prostředí ROS (Robot Operating System).
2. Seznamte se s řídicími jednotkami robotů MITSUBISHI RV6-S a RV6-SL. Pozornost věnujte především možnostem připojení a jejich komunikačnímu protokolu.
3. Vyjděte z existujících řešení a realizujte pro uvedené manipulátory základní programové rozhraní v prostředí ROS. Jeho součástí bude kolizní a vizualizační model. Rozhraní koncipujte a realizujte s ohledem na použití při výuce.
4. V programovém rozhraní implementujte také ovládání chapadla.
5. Realizujte analytický výpočet inversní kinematické úlohy pro uvedené manipulátory.
6. Připravte vývojovou a uživatelskou dokumentaci. Součástí dokumentace musí být i detailní návod pro instalaci a použití studenty při výuce.

Seznam doporučené literatury:

[1] Haruhiko Asada, Jean-Jacques Slotine: Robot Analysis and Control. John Wiley and Son, New York, USA 1986, ISBN: 978-0471830290
[2] Reza N. Jazar: Theory of Applied Robotics: Kinematics, Dynamics, and Control. Springer, 2010, ISBN: 978-1441917492
[3] MITSUBISHI: Manuály robotů RV6-S a RV6-SL (dodá vedoucí)
[4] MoveIt: MoveIt tutorials, http://docs.ros.org/en/melodic/api/moveit_tutorials/html/index.html

Jméno a pracoviště vedoucí(ho) bakalářské práce:

**Ing. Pavel Krsek, Ph.D., robotické vnímání CIIRC**

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **05.01.2021** Termín odevzdání bakalářské práce: **21.05.2021**

Platnost zadání bakalářské práce: **30.09.2022**

| Ing. Pavel Krsek, Ph.D. | prof. Ing. Tomáš Svoboda, Ph.D. | prof. Mgr. Petr Páta, Ph.D. |
|---|---|---|
| podpis vedoucí(ho) práce | podpis vedoucí(ho) ústavu/katedry | podpis děkana(ky) |

## III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. | Datum převzetí zadání

Podpis studenta

# Poděkování

Chtěl bych poděkovat svému vedoucímu Ing. Pavlu Krskovi, Ph.D. za jeho trpělivost s mým nechápavým výrazem a za odpovědi na emaily i v pozdních hodinách. Své rodině (kocoura nevyjímaje) děkuji za podporu a jejich otevřené uši, když jsem si potřeboval utřídit myšlenky. Dále děkuji Mgr. Romanu Sejkotovi za pořízení fotografií robotů.


# Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 21. května 2021

# Abstrakt

Tématem této práce je vytvoření ovládacího rozhraní pro roboty firmy Mitsubishi v prostředí ROS (Robotic operating system). V projektu jsou využity manipulátory RV-6SDL a RV-6S s příslušnými řídícími jednotkami. Robot RV-6SDL je vybaven chapadlem PG 70 od firmy Schunk. Pro chapadlo PG 70 je také vytvořeno ovládací rozhraní. Vzniklá robotická pracoviště jsou plánována k využití při výuce robotiky na ČVUT Fakultě elektrotechnické. Rozhraní jsou navrhována s ohledem na výuku. Další částí projektu je implementace vlastního analytického řešení přímé a inverzní kinematické úlohy. Pro ulehčení práce s robotickými buňkami je pro studenty vytvořen modul v jazyce Python 2.7. Tento modul předkládá uživateli funkce k ovládání robotu. Práce je doplněna o dokumentaci.



**Klíčová slova:** ROS, Ovládací rozhraní, Mitsubishi, RV-6S, RV-6SDL, Schunk, PG70

**Školitel:** Ing. Pavel Krsek, Ph.D.
ČVUT CIIRC



# Abstract

The aim of this thesis is to create control interface for Mitsubishi robots in ROS (Robotic Operating System) environment. The RV-6SDL and RV-6S robotic manipulators were used in this project along with their corresponding control units. Robot RV-6SDL is equipped with the PG 70 gripper from the Schnunk company. Control interface for gripper PG 70 was also created. These robotic workplaces are to be used for teaching robotics classes at Faculty of electrical engineering at CTU. Interfaces are designed with consideration to the educational character of application. In the second part of the project the analytic kinematics solver is implemented. A module in Python 2.7 has been created for simplification of use. This module provides the user with functions for robot control. The documentation has been also created.



**Keywords:** ROS, Control interface, Mitsubishi, RV-6S, RV-6SDL, Schunk, PG70

**Title translation:** Control interface for MITSUBISHI robots in ROS

# Obsah

# Kapitola 1
# Úvod

Budoucnost průmyslu patří robotům. Již několik let se lidstvo snaží přenést část svých povinností na stroje, které by je uskutečnily. V posledních několika desítkách let se tato snaha dostala do bodu, kdy již dokážeme vyrábět a řídit stroje, jako jsou průmyslové manipulátory, autonomní vozidla a jiné.
I když dnes zaznamenáváme rychlý rozvoj v oblastech neuronových sítí a umělé inteligence, je přesto důležité, předávat dalším generacím inženýrů znalosti o kinematice a dynamice robotických manipulátorů.
Na Fakultě elektrotechnické ČVUT se proto vyučují předměty s touto tématikou. Studenti mají možnost plnění praktických úloh na pracovištích vybavených robotickými manipulátory různých typů.
Tato práce je součásti snahy zařadit do výuky další dvě pracoviště s průmyslovými manipulátory. Pracoviště jsou určena především pro studentskou práci v rámci výuky.
Nové robotické buňky jsou připravovány pro roboty Mitsubishi. Jedná se o sériové manipulátory RV-6S a RV-6SDL. Tyto roboty nepatří k nejnovější generaci. Přesto jsme přesvědčeni, že budou vyhovovat účelům výuky.
Roboty jsou řízeny pomocí řídících jednotek připojených k počítači s prostředím ROS (Robotic Operating system). Aby bylo možné roboty zprovoznit, musí být vytvořeno rozhraní mezi systémem ROS a řídící jednotkou robotu. Pro snadnou integraci do systému knihoven prostředí ROS musí rozhraní umožňovat řízení robotu po trajektorii.
Součástí této práce je také vytvoření vizualizačních a kolizních modelů robotů. Tyto modely spolu s omezeními v řídících jednotkách robotů zajišťují bezpečnost práce.
Rozhraní využívá analytické řešení inverzní kinematické úlohy. Řešení kinematických úloh pro použité roboty je v této práci odvozeno a implementováno v jazyce Python. Vzniklý algoritmus je integrován do systému ROS.
Dále je vypracována demonstrační úloha, která ukazuje funkčnost pracoviště. Obdobná úloha by mohla být součástí praktických cvičení při výuce. K vypracování úlohy je použit modul v jazyce Python, který usnadňuje práci s roboty a který je také součástí této práce.
Na závěr jsou vytvořeny dokumentace pracovišť a modulů v jazyce Python určených pro uživatele. Tyto dokumenty jsou psány v angličtině pro snazší šiřitelnost.

# Kapitola 2

# Základní pojmy a formalismus

## 2.1 Matematický formalismus

Jednotlivé složky bodů a vektorů budu značit následujícím způsobem

$$\boldsymbol{P} = \begin{bmatrix} P_x \\ P_y \\ P_z \end{bmatrix}$$

Pro účely výpočtů kinematických úloh budou potřeba matematické interpretace prostorové rotace a translace. Pro zjednodušení budou využity homogenní souřadnice a pro ně vhodné transformace.[1]
V této práci se budu přidržovat následujícího formalismu. Pokud není uvedeno jinak, platí:

- matice $R_X(\alpha)$ značí rotační matici kolem osy $X$ o úhel $\alpha$. Ekvivalentně platí i pro ostatní souřadné osy.

$$R_X(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\alpha) & -sin(\alpha) \\ 0 & sin(\alpha) & cos(\alpha) \end{bmatrix}, \tag{2.1}$$

$$R_Y(\alpha) = \begin{bmatrix} cos(\alpha) & 0 & sin(\alpha) \\ 0 & 1 & 0 \\ -sin(\alpha) & 0 & cos(\alpha) \end{bmatrix}, \tag{2.2}$$

$$R_Z(\alpha) = \begin{bmatrix} cos(\alpha) & -sin(\alpha) & 0 \\ sin(\alpha) & cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.3}$$

- matice $R$ značí obecnou rotační matici o rozměrech $3 \times 3$. Skládá se z libovolné kombinace rotací kolem souřadných os.
- matice $t(\boldsymbol{v})$ značí translaci ve směru vektoru $\boldsymbol{v}$ o vzdálenost $|\boldsymbol{v}|$ pro homogenní souřadnice.

$$t(\boldsymbol{v}) = \begin{bmatrix} 1 & 0 & 0 & v_x \\ 0 & 1 & 0 & v_y \\ 0 & 0 & 1 & v_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.4}$$

- matice $T$ značí transformační matici $4 \times 4$ pro homogenní souřadnice.

$$T = \begin{bmatrix} & & & v_x \\ & R & & v_y \\ & & & v_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.5}$$

Dále zavedu značení transformací mezi systémy a označení bodů podle souřadného systému.

- $\boldsymbol{P}^{\boldsymbol{a}}$ značí bod $\boldsymbol{P}$ ve vztažné soustavě $a$.

- $T_b^a$ značí transformaci souřadnic ze systému $a$ do systému $b$. Stejné značení platí i pro rotační matice.

Nastávají případy, kdy je potřeba ve výpočtech přistupovat k jednotlivým prvkům matic. Číslování bude probíhat od jedničky. Indexování bude mít formát, kdy

$$R[1, 3],$$

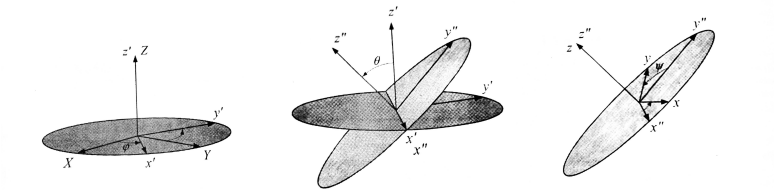
odkazuje na prvek v prvním řádku a třetím sloupci matice $R$.

## 2.2 Kinematika manipulátorů

Kinematika je věda, zabývající se geometrií pohybu robotu a trajektorie, po kterých se pohybuje mezi jednotlivými body.

Nejprve je třeba definovat počet stupnů volnosti (dále DOF[1]). Jedná se o minimální počet parametrů potřebných k úplnému popsání tělesa nebo skupiny těles v daném prostoru. Pokud se obecné těleso nachází v tří-dimenzionálním prostoru, má celkem šest DOF. Tři stupně volnosti představuje pozice v prostoru, popsaná například kartézskými souřadnicemi a druhé tři určují natočení. Rotaci je možné reprezentovat několika způsoby. Například nepraktickou rotační maticí, neintuitivní reprezenzací osa-úhel nebo pomocí kvaternionu.
V této práci budu používat reprezentaci pomocí Eulerových úhlů. Tato varianta má své nevýhody, ale jedná se o interpretaci používanou v hodinách robotiky. Z výukových důvodů zachovávám použití Eulerových úhlů.
Eulerovy úhly jsou různě definovány. V některých odvětvích se využívá varianta Yaw, Pitch, Roll. Já budu využívat variantu označovanou $ZXZ$, která je znázorněna na Obrázku 2.1.

[1] Degrees of freedom



**Obrázek 2.1:** Ilustrace reprezentace rotace v prostoru Eulerovými úhly [1]

V případě použité varianty Eulerových úhlů je rotační matice tvořena složením dle 2.6.

$$R\left(\phi,\theta,\psi\right) = R_Z(\phi)R_{X'}(\theta)R_{Z''}(\psi) \tag{2.6}$$

Polohový vektor v $\mathbf{R}^3$ má pak tvar

$$X = \left[X_x, X_y, X_z, X_\phi, X_\theta, X_\psi\right]^T. \tag{2.7}$$

Rozlišujeme přímou a inverzní kinematickou úlohu.

- **Přímá kinematická úloha** je zobrazení

$$\boldsymbol{J} \rightarrow \boldsymbol{X}, \text{ kde} \tag{2.8}$$
$$\boldsymbol{J} \in \mathbf{R}^n, \tag{2.9}$$
$$\boldsymbol{X} \in \mathbf{R}^6. \tag{2.10}$$

  Vektor $\boldsymbol{J}$ je známá kloubová souřadnice[2] a $n$ je počet řízených kloubů manipulátoru. V případě použitých manipulátorů $n = 6$. Vektor $\boldsymbol{X}$ je poloha v prostoru podle 2.7.

- **Inverzní kinematická úloha** je zobrazení

$$\boldsymbol{X} \rightarrow \boldsymbol{J}, \text{ kde} \tag{2.11}$$
$$\boldsymbol{J} \in \mathbf{R}^n, \tag{2.12}$$
$$\boldsymbol{X} \in \mathbf{R}^6. \tag{2.13}$$

  Vektor $\boldsymbol{X}$ je známá poloha v prostoru podle 2.7 a vektor $\boldsymbol{J}$ představuje hledanou kloubovou souřadnici. Parametr $n = 6$.

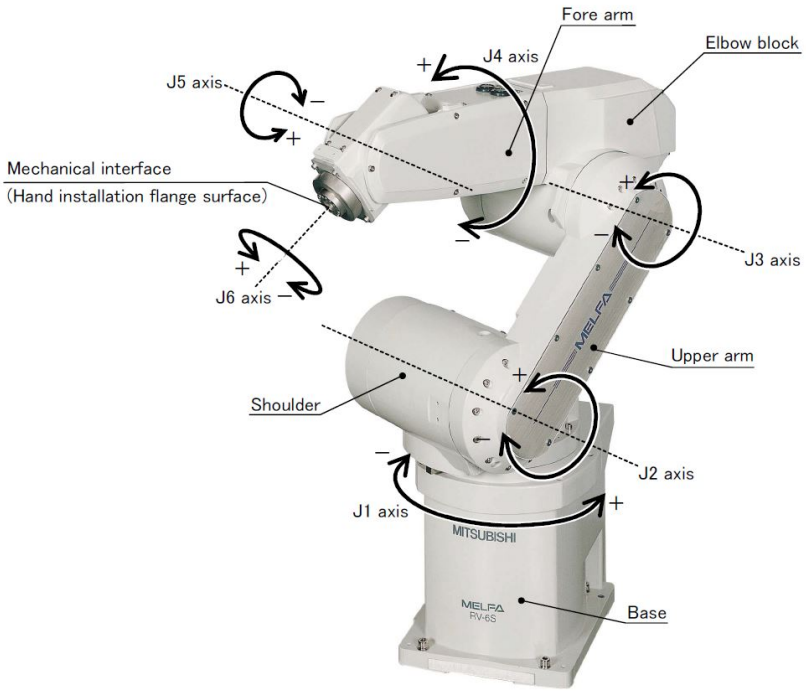
[2]Kloubová souřadnice je vektor hodnot řízených kloubů.

# Kapitola 3

# Technické a programové prostředky

## 3.1 Použité roboty a řídící jednotky

Roboty využívané v tomto projektu patří do třídy sériových manipulátorů s otevřeným kinematickým řetězcem a šesti stupni volnosti (dále 6DOF). Jednotlivé roboty RV-6S a RV-6SDL se od sebe liší na první pohled pouze ve velikostech některých ramen. V dokumentaci poskytnuté od Mitsubishi je možné se dočíst, že RV-6S dokáže vyvinout vyšší rychlost v kloubech J1, J2 a J3 (viz Obrázek 3.1). Je tomu tak nejspíše díky jeho menším rozměrům a s tím spojenou menší hmotností mechanismu. [2] [3]




**Obrázek 3.1:** Robot RV-6S[2]

Oba roboty jsou ovládány řídícími jednotkami. Ty se starají o regulaci a komunikaci. Dále je možné využít univerzální vstupní a výstupní porty

ke snímání a spínání dalších periferií. Samotné manipulátory obsahují pouze motory, snímače polohy, hadičky pro případné pneumatické nástroje a přídavnou kabeláž pro rozšiřování.
Manipulátor RV-6SDL je řízen jednotkou CR1D-700 a RV-6S jednotkou CR2A-572. Řídicí jednotky (kontroléry) nejsou, bohužel i přes podobnost manipulátorů, rovnocenné. To komplikuje přípravu programového rozhraní. Jednotka CR1D-700 (pro robot RV-6SDL) je vybavena, na rozdíl od CR2A-572, rozhraním Ethernet 100Base-T. Dále CR1D-700 podporuje vyšší verzi programovacího jazyka MELFABASIC (verzi V). Tyto dva aspekty vedou k možnosti využití funkcionality `MXT control`. Jedná se o rozhraní využívající Ethernet k ovládání robotu v reálném čase. Díky tomu je možné pro robot RV-6SDL poměrně snadno řídit trajektorii.[4]
Naproti tomu jednotka CR2A-572 žádné takové funkce nenabízí. Podporuje pouze MELFABASIC IV a je vybavena pouze rozhraním RS-232 (které obsahuje i jednotka CR1D-700). Tento systém tedy neumožňuje žádné externí řízení v reálném čase. Bude třeba jej vytvořit nebo robot řídit pouze pozičně.[5]
Obě jednotky podporují komunikační protokol společnosti Mitsubishi. Výrobce neuvádí jeho jméno. Budu ho označovat jako `Melfa protokol`. Protokol je textového charakteru. Tento protokol je možné provozovat přes Ethernet i RS-232. Protokol slouží převážně k diagnostice, nastavování parametrů, zapisování zdrojového kódu, spouštění a přerušování programů. Mimo to umožňuje získávání informací o současné poloze koncového bodu robotu a zadání pohybu na jinou pozici.[6]


**Obrázek 3.2:** Chapadlo Schunk PG 70[7]

Robot RV-6SDL byl vybaven chapadlem PG 70 od firmy Schunk. Je upevněno na mechanickém rozhraní (na Obrázku 3.1 "mechanical interface"). Jedná se o dvouprsté paralelní chapadlo řízené pomocí RS-232. Tento nástroj disponuje svým vlastním komunikačním protokolem. Mimo řízení polohy prstů, je možné nastavit také rychlost pohybu a maximální proud vinutím motoru. Přiložené CD obsahuje fotografie robotických pracovišť.

## 3.2 ROS - Robotic Operating System

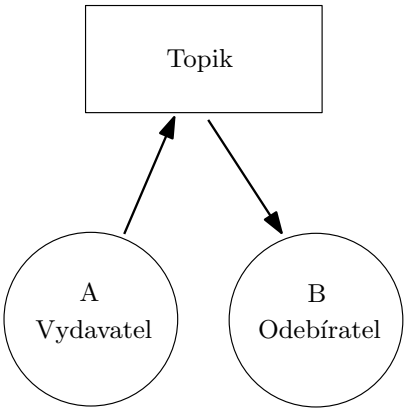
Vývoj robotických aplikací se může snadno stát velmi komplikovaným. Je třeba zajistit, aby dané periferie spolu řádně komunikovaly, aby aktuátory (motory, pneumatické obvody) reagovaly ve správnou chvíli a podobně. Za účelem zjednodušení těchto potíží byl vyvinut systém ROS, operační systém pro robotiku. Jedná se o souhrn knihoven ulehčujících komunikaci periferií, řízení a plánování pohybu. Byl vytvořen spolupracujícím kolektivem vývojářů v oblasti robotiky. Jedná se o neziskový open source projekt.
Systém ROS pro svou činnost potřebuje operační systém počítače. Obvykle se používá Linux, ale je možné jej instalovat i na platformu Windows s většími či menšími úspěchy. Zdrojové kódy jsou psány v programovacím jazyce C/C++, ale je možné také psát skripty v jazyce Python 2.7 (verze Python 3 není v použité verzi ROS podporována).[15]

## 3.3 Struktura systému ROS

Systém ROS se celý skládá z balíčků (packages). Tyto balíčky obsahují jednotlivé části projektů. Může se jednat například o balíček obsahující popis robotu, jeho vizuální a kolizní model, nebo může obsahovat rozsáhlou knihovnu, jakou je například balíček MoveIt. Systém balíčků slouží hlavně pro přehlednou organizaci projektu. Každý balíček je obsažen v jednom adresáři, ale každý projekt může využívat i balíčky mimo svůj adresář. Každý balíček obsahuje soubor `package.xml`, ve kterém jsou definovány závislosti balíčku na ostatních balíčcích, exportování knihoven a modulů. Dále každý balíček obsahuje soubor `CMakeList.txt`, který nastavuje kompilaci balíčku, zdrojových kódů a jejich sestavování ve spustitelné soubory.[15]
Základem každého ROS projektu je takzvaný workspace. Jedná se o adresář obsahující alespoň složku `src`, který byl pomocí příkazu `catkin init` inicializován jako ROS workspace. Složka `src` obsahuje nově vytvářené balíčky pro danou aplikaci. Před spuštěním vytvořených programů je nutné workspace zkompilovat pomocí příkazu `catkin build`.[15]
Používání systému ROS je možné si ulehčit speciálními příkazy pro příkazový řádek. Například příkaz `roscd` umožňuje přesouvání mezi balíčky, `rosed` pohodlnější editaci souborů a `rospack` inspekci balíčků, jejich závislostí, jaké poskytují knihovny a podobně.
Dále systém ROS nabízí vizualizační program Rviz a fyzikální simulátor Gazeebo. Program Rviz se často používá pro vizualizaci i při řízení fyzického robotu.[15]
Další často využívaná funkcionalita je parameter server. Ten slouží jako databáze parametrů a jiných informací. Typicky se na něj nahrávají informace jako model robotu, rozsahy kloubů a nebo nastavení komunikačních periferií.

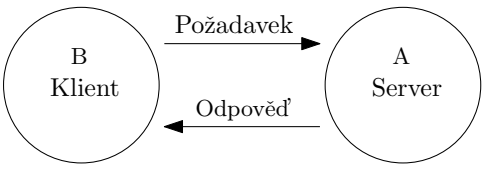
## 3.4 Uzel, topik, zpráva, služba a klient

Každá ROS aplikace se skládá z uzlů (nodes). Každý uzel představuje jeden program. Uzel je možné programovat v jazyce C/C++ a nebo Python. Aby uzel a celý systém správně fungoval, musí být spuštěn program `roscore`. Uzel je možné spustit buď stejně jako normální spustitelný soubor nebo skript a nebo pomocí příkazu `rosrun`. Pokud je potřeba spustit větší počet uzlů, s výhodou se využije soubor typu `.launch` a příkaz `roslaunch`.
V souboru typu `.launch` je definováno, jaké uzly z jakých balíčků je třeba spustit. V tomto případě již není potřeba spouštět manuálně program `roscore`, učiní se tak automaticky. Pomocí takového souboru je také možno snadno nahrávat data na parameter server.




**Obrázek 3.3:** Znázornění komunikace dvou uzlů pomocí topiku

Uzly mezi sebou komunikují přes takzvané topiky [1]. Uzel A do topiku zapíše zprávu (message), takovému uzlu se říká vydavatel (publisher), a uzel B si zprávu přečte, říká se mu proto odebíratel (subscriber). Graficky je komunikace znázorněna na Obrázku 3.3.
Zpráva má předem danou strukturu, která je definována pomocí souboru s příponou `.msg` umístěného v rámci balíčku. Pro její použití je nutné balíček kompilovat a tím je struktura zprávy publikována i mezi ostatní balíčky.
Zprávy nejsou principiálně čteny periodicky a ani není generováno přerušení při změně v topiku. Odebíratel musí zajistit dostatečně pravidelné čtení topiku.
Typickým příkladem využití komunikace Uzel-Topik-Uzel je publikování kloubových souřadnic robotu (topik `/joint_states`).




**Obrázek 3.4:** Příklad komunikace dvou uzlů jako server a klient

Druhým základním způsobem komunikace je pomocí systému server-klient. Uzel A poskytuje službu (service) a je označován jako server. Uzel B požaduje poskytnutí služby, říká se mu klient. Forma služby je definována obdobě jako zpráva, souborem s příponou `.srv`. V tomto konfiguračním souboru je definována struktura požadavku a odpovědi. Pro správné propagování formátu je nutná kompilace.
Server A může být samozřejmě ve chvíli, kdy je volán klientem B, vytížen jinými klienty a neodpoví. V takovém případě se musí klient přizpůsobit.

[1] Z anglického topic. V tomto příkladě mi český překlad 'téma' přišel nevhodný a proto používám počeštěnou variantu anglického termínu.

Typickým příkladem tohoto typu komunikace může být výpočet inverzní kinematické úlohy nebo kontrola validity stavu robotu.[15]
Někdy je možné se setkat s pojmem akční server (action server). V principu se jedná o podobný typ komunikace jako s klasickým serverem, s tím rozdílem, že požadavek obnáší provedení fyzické akce. Například se může jednat o nastavení pozice kloubu robotu. Dále akční server umožňuje zavést zpětnou vazbu pomocí topiku feedback. Jedná se o rozšíření dodané balíčkem actionlib.

## 3.5 MoveIt

MoveIt je velmi rozsáhlý balíček obsahující podporu plánování a řízení robotů. Vytváří komplexní subsystém zajišťující spouštění všech potřebných služeb, kontrolérů a vizualizace. Zahrnuje knihovnu OMPL (Open Motion Planning Library) pro plánování pohybu. Tato knihovna zajišťuje plánování trajektorie dané kartézskými souřadnicemi. Obsahuje několik algoritmů časové parametrizace, mezi kterými může uživatel volit pomocí konfiguračního souboru.
V základní konfiguraci MoveIt nabízí i řešení inverzní kinematické úlohy numerickou metodou. Výhodou takového postupu je univerzálnost algoritmu. Naproti tomu jsou numerické metody obvykle méně stabilní než analytické. MoveIt však nabízí zavedení vlastního řešení inverzní kinematické úlohy ať už pomocí služby a nebo celým vlastním kinematickým modulem.

## 3.6 ROS control

Další potřebnou funkcionalitou je vytvoření rozhraní mezi robotem a systémem ROS. Za tímto účelem je využit balíček ROS control. Ten nabízí řízení pozice, rychlosti, momentu a trajektorie. V tomto projektu byly využívány poziční kontroléry trajektorie a poziční ovládání chapadla. Pro funkci kontrolérů jsou využity akční servery. Blíže se budu tématu věnovat v kapitolách 5.4 a 5.5.

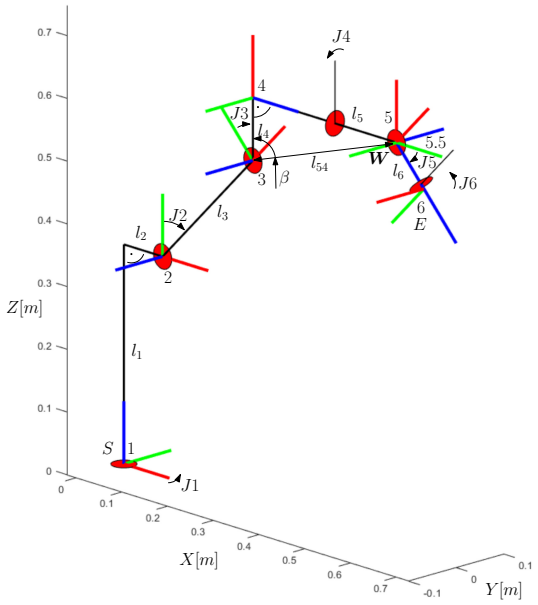
# Kapitola 4

# Řešení přímé a inverzní kinematické úlohy

Balíček MoveIt poskytuje svou numerickou variantu řešení inverzní kinematické úlohy. Tento způsob výpočtu není však úplně vhodný. Doba vyhodnocení je závislá na požadované přesnosti. Dále je možné, že výsledek nebude ke správnému řešení ani konvergovat. Tento případ jsem nepozoroval, ale není vyloučené, že by mohl nastat. Z těchto důvodů jsem vytvořil vlastní analytické řešení inverzní kinematické úlohy opírající se o geometrická pozorování.
Ve všech doprovodných schématech se držím konvence, kdy osy X, Y, Z systémů mají barvu červenou, zelenou, modrou a jsou vyvedeny pouze v kladném směru. Rotační klouby jsou značeny červeným kolečkem, přičemž normála k rovině kolečka je shodná s osou rotace kloubu.

## 4.1 Odvození řešení přímé kinematické úlohy

Vyšetřované roboty můžeme znázornit pomocí schématu na Obrázku 4.1.




**Obrázek 4.1:** Schématický nákres robotu

Je dána rovnice

$$\boldsymbol{P^S} = T_E^S \boldsymbol{P^E} = T_1^B T_2^1 T_3^2 T_4^3 T_5^4 T_{5.5}^5 T_6^{5.5} T_E^6 \boldsymbol{P^E}. \tag{4.1}$$

Pokud jsou do 4.1 dosazeny za $\boldsymbol{P^E}$ souřadnice počátku v homogenních souřadnicích, vznikne vzorec pro pozici počátku systému koncového bodu manipulátoru[1]. Díky faktu, že světový systém $S$ je shodný se systémem 1, není nutné vyšetřovat $T_S^1$. Transformace $T_E^6$ musí být zadána spolu s délkami ramen a mezními rozsahy kloubů.
Pomocí Denavit-Hartengergovy notace (dále DH notace) je vytvořen popis robotu pro nalezení transformací mezi systémy. Tato notace umožňuje popsat transformace mezi souřadnicovými systémy kloubů pouze čtyřmi parametry.[1]

| $T$ | $\theta$ [rad] | $d$ [mm] | $a$ [mm] | $\alpha$ [rad] |
|---|---|---|---|---|
| $1 \rightarrow 2$ | $J1$ | $l_1$ | $l_2$ | $\frac{\pi}{2}$ |
| $2 \rightarrow 3$ | $\frac{\pi}{2} - J2$ | 0 | $l_3$ | 0 |
| $3 \rightarrow 4$ | $\frac{\pi}{2} - J3$ | 0 | $l_4$ | $\frac{\pi}{2}$ |
| $4 \rightarrow 5$ | $J4$ | $l_5$ | 0 | $\frac{\pi}{2}$ |
| $5 \rightarrow 5.5$ | $J5$ | 0 | 0 | $-\frac{\pi}{2}$ |
| $5.5 \rightarrow 6$ | $J6 + \frac{\pi}{2}$ | $l_6$ | 0 | 0 |

**Tabulka 4.1:** Parametry Denavit-Hartenbergovy notace pro roboty Mitsubishi

Po dosazení dat z Tabulky 4.1 a koncové transformace do rovnice 4.1 vznikne matice $T_E^S$ závislá pouze na kloubových souřadnicích. Pozice koncového bodu je dána jako

$$\boldsymbol{P^S} = \begin{bmatrix} X_x \\ X_y \\ X_z \\ 1 \end{bmatrix} = T_E^S \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}. \tag{4.2}$$

Orientace koncového systému je získána jako řešení rovnice

$$R_Z(X_\phi) R_{X'}(X_\theta) R_{Z''}(X_\psi) = R_T, \tag{4.3}$$

kde $R_T$ je rotační část transformační matice $T_E^S$.

## 4.2 Odvození řešení inverzní kinematické úlohy

Při odvozování je využit fakt, že v případě použitých robotů se osy kloubů J4, J5, J6 protínají v jednom bodě. Tento bod je označen jako $\boldsymbol{W}$. To umožňuje řešit nejprve inverzní kinematickou úlohu pro spodní tři klouby J1, J2, J3 a polohu bodu $\boldsymbol{W}$ (bez orientace). Pak je možné využít postup, kdy je zaveden dodatečný systém k již vytvořeným systémům. Tento systém má počátek v bodě $\boldsymbol{W}$, má stejnou orientaci jako systém 6 a je označen číslem 7.

[1] Bez orientace.

### 4.2.1 Odvození řešení inverzní kinematické úlohy pro první tři klouby

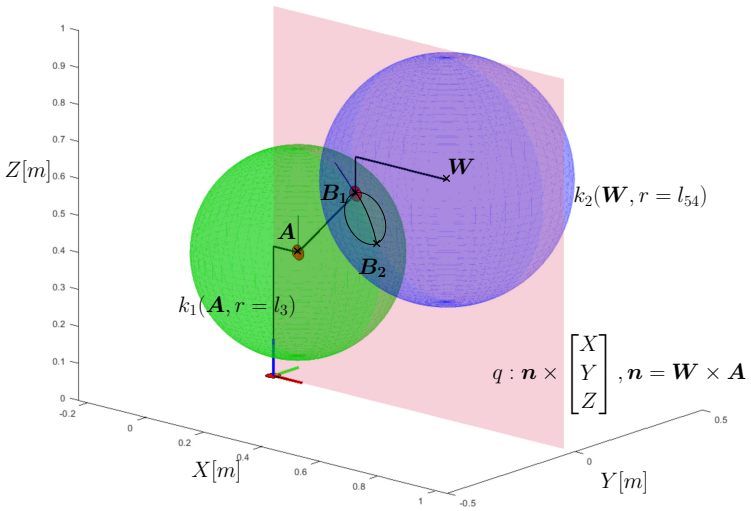
Bod $\boldsymbol{W}$ je určen jako

$$\boldsymbol{W} = T_6^S t_7^6 \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}, \tag{4.4}$$

$$t_7^6 = t\left(\begin{bmatrix} 0 \\ 0 \\ -l_6 \end{bmatrix}\right), \tag{4.5}$$

$$T_6^S = t\left(\begin{bmatrix} X_x \\ X_y \\ X_z \end{bmatrix}\right) R_Z\left(X_\phi\right) R_X\left(X_\psi\right) R_Z\left(X_\theta\right) T_E^6, \tag{4.6}$$

kde matice $R_Z$ a $R_X$ jsou rotační matice $4 \times 4$ pro transformaci homogenních souřadnic. Vektor $\boldsymbol{X}$ představuje zadanou polohu a orientaci koncového efektoru podle formátu 2.7.
Dále je nalezeno řešení inverzní kinematické úlohy pro spodní tři klouby a bod $\boldsymbol{W}$. Body v prostoru, kde se klouby nacházejí, jsou označeny velkými písmeny. Nejprve jsou nalezeny tyto body a poté úhly mezi rameny robotu.




**Obrázek 4.2:** Schématické zakreslení spodních tří kloubů manipulátoru doplněné o geometrická pozorování

Bod $\boldsymbol{A}$ má fixní vzdálenost od počátku na ose $Z$ a zbývá určit souřadnice $X$ a $Y$. Kloub J1 je také z těchto tří jediný, který dokáže zajistit rotaci

manipulátoru kolem osy $Z$ systému $S$. Platí

$$\boldsymbol{A}_{1,2} = \begin{bmatrix} 0 \\ 0 \\ l_1 \end{bmatrix} \pm l_2 \frac{\boldsymbol{W_{XY}}}{|\boldsymbol{W_{XY}}|}, \tag{4.7}$$

kde $\boldsymbol{W_{XY}}$ je kolmou projekcí bodu $\boldsymbol{W}$ do roviny $XY$.
Dále pro přehlednost zavedeme:

$$l_{54} = \sqrt{l_4^2 + l_5^2} \tag{4.8}$$

$$d = |\boldsymbol{W} - \boldsymbol{A}| \tag{4.9}$$

Podle $d$ se následně odvíjí počet řešení pro bod $\boldsymbol{B}$. Pokud $d$, $l_{54}$ a $l_3$ nesplňují trojúhelníkovou nerovnost, nemá úloha řešení.
Dále pokud platí

$$|d - (l_3 + l_{54})| = 0, \tag{4.10}$$

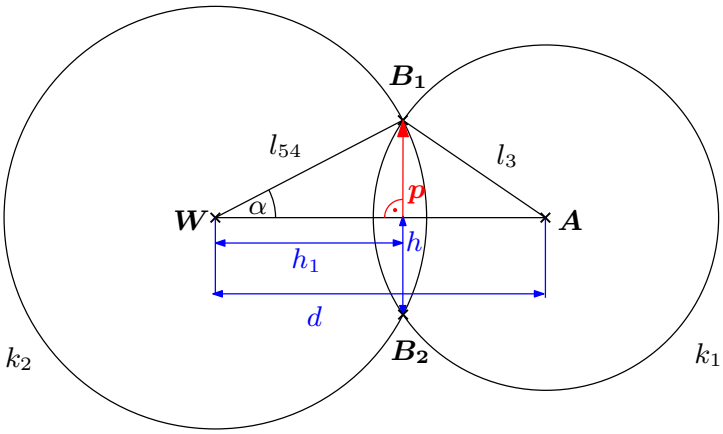
má úloha pro bod $\boldsymbol{B}$ jedno řešení

$$\boldsymbol{B} = \boldsymbol{W} + \frac{\boldsymbol{W} - \boldsymbol{A}}{|\boldsymbol{W} - \boldsymbol{A}|} l_{54}. \tag{4.11}$$

V ostatních případech, kdy

$$d < l_3 + l_{54}, \tag{4.12}$$

má inverzní kinematická úloha pro bod $\boldsymbol{B}$ dvě řešení. Na Obrázku 4.2 je vidět, že bod $\boldsymbol{B}$ leží na průsečících dvou koulí $k_1$ a $k_2$ a roviny $q$. Díky symetrii koulí lze úlohu zjednodušit na hledání průsečíku dvou kružnic.




**Obrázek 4.3:** Ilustrace situace pro hledání průsečíků dvou kružnic

Z Obrázku 4.3 vyplývá

$$cos(\alpha) = \frac{l_{54}^2 + d^2 - l_3^2}{2l_{54}d}, \tag{4.13}$$

$$h = l_{54}sin(\alpha), \tag{4.14}$$

$$h_1 = l_{54}cos(\alpha), \tag{4.15}$$

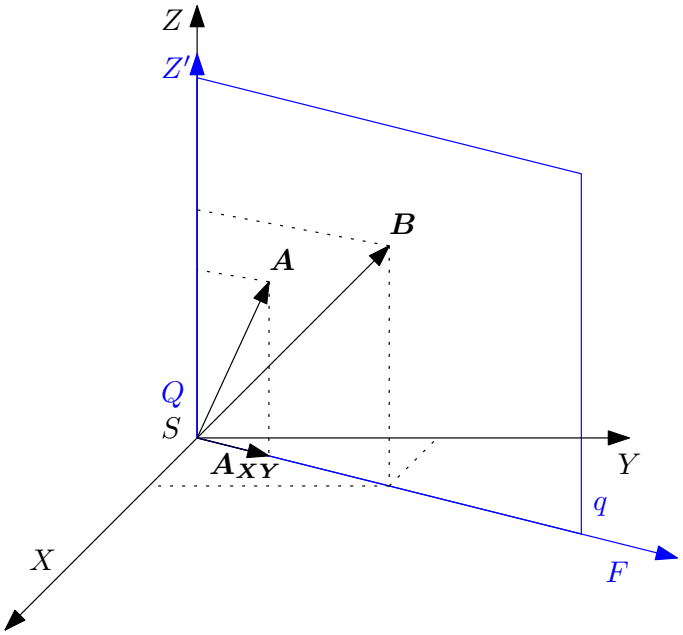
$$\boldsymbol{p} = \left( \boldsymbol{A} \times \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \right) \times (\boldsymbol{W} - \boldsymbol{A}), \tag{4.16}$$

$$\boldsymbol{B_{1,2}} = \boldsymbol{W} + h_1 \frac{\boldsymbol{W} - \boldsymbol{A}}{|\boldsymbol{W} - \boldsymbol{A}|} \pm h \frac{\boldsymbol{p}}{|\boldsymbol{p}|}. \tag{4.17}$$

Tím jsou určeny všechny potřebné body v prostoru a zbývá pouze určit úhly mezi rameny, kloubové souřadnice.

$$J1 = arctan2\,(A_y, A_x) \tag{4.18}$$

Vyšetřování kloubových souřadnic $J2$ a $J3$ je zjednodušeno projekcí, při níž jsou souřadnice bodů $\boldsymbol{A}$, $\boldsymbol{B}$ a $\boldsymbol{W}$ ze systému $S$ převedeny do dvoudimenzionálního sytému $Q$.




**Obrázek 4.4:** Ilustrace převodu do systému $Q$

Počátek systému $Q$ je totožný se systémem $S$. Souřadné osy $Z$ a $Z'$ jsou totožné, a souřadná osa $F$ tohoto systému má kladný směr daný vektorem $\boldsymbol{A_{XY}}$.
Transformovaný bod $\boldsymbol{G}$ je označen jako $\boldsymbol{G'}$. Jelikož všechny převáděné body

již v rovině $q$ leží, lze transformaci z $S$ do $Q$ psát jako

$$G'_f = \begin{cases} \sqrt{G_x^2 + G_y^2}, & \text{pokud } \frac{G_{XY}}{|G_{XY}|} = \frac{A_{XY}}{|A_{XY}|} \\ -\sqrt{G_x^2 + G_y^2}, & \text{pokud } \frac{G_{XY}}{|G_{XY}|} \neq \frac{A_{XY}}{|A_{XY}|} \end{cases}, \tag{4.19}$$

$$G'_{z'} = G_z. \tag{4.20}$$

Zbývající kloubové souřadnice jsou určeny jako

$$\boldsymbol{AB'} = \boldsymbol{A'} - \boldsymbol{B'}, \tag{4.21}$$

$$J2 = \frac{\pi}{2} - arctan2(AB'_{z'}, AB'_f). \tag{4.22}$$

Pokud hodnota $J2$ vyjde mimo interval $\langle -\pi, \pi \rangle$ je do tohoto intervalu převedena vztahem

$$\alpha = \begin{cases} -2\pi + \alpha, & \text{pokud } \alpha > \pi \\ 2\pi + \alpha, & \text{pokud } \alpha < -\pi \end{cases}. \tag{4.23}$$

Dále

$$\boldsymbol{BW'} = \boldsymbol{B'} - \boldsymbol{W'}, \tag{4.24}$$

$$\beta = arccos\left(\frac{l_4}{l_{54}}\right), \tag{4.25}$$

$$J3 = \pi sign\,(J2) - J2 - arctan(BW'_{z'}, BW'_f) - \beta. \tag{4.26}$$

Úhel $J3$ je také případně převeden do intervalu $\langle -\pi, \pi \rangle$ pomocí vztahu 4.23. Tím je vyřešena inverzní kinematika pro první tři klouby manipulátoru. Byly nalezeny pouze základní úhly. Pro robot s nekonečnými rozsahy kloubů je řešení nekonečně mnoho lišících se o $2k\pi$, kde $k \in \mathbf{Z}$.

### 4.2.2 Odvození řešení inverzní kinematické úlohy pro druhé tři klouby

Následujícím postupem jsou pro každé nalezené řešení pro klouby $J1$, $J2$ a $J3$ spočteny kloubové souřadnice $J4$, $J5$ a $J6$. Z dříve nalezené DH notace a znalosti hodnot kloubů $J1$, $J2$ a $J3$ je získána matici $R_4^S$. Z rovnice 4.27 je zjištěna požadovaná matice $R_4^6$.

$$R_4^6\,(J4, J5, J6) = \left(R_4^S\right)^T R_6^S \tag{4.27}$$

Z DH notace plyne, že

$$R_4^6\,(J4, J5, J6) = \begin{bmatrix} -c(J4)c(J5)s(J6) & -c(J4)c(J5)c(J6) + s(J4)s(J5) & -c(J4)s(J5) \\ s(J4)c(J5)s(J6) & -s(J4)c(J5)c(J6) - c(J4)s(J6) & -s(J4)s(J5) \\ -s(J5)s(J6) & -s(J5)c(J6) & c(J5) \end{bmatrix},$$

kde funkce $c(\alpha) = cos(\alpha)$ a $s(\alpha) = sin(\alpha)$. Rovnice má až dvě řešení, která lze získat pomocí následujícího postupu.
Pokud $R_4^6[3,3] \in (-1;1)$

$$J5_{1,2} = arccos\left(R_4^6[3,3]\right), \tag{4.28}$$

$$J6_{1,2} = arctan2\left(-sin(J5_{1,2})R_4^6[3,1], -sin(J5_{1,2})R_4^6[3,2]\right), \tag{4.29}$$

$$J4_{1,2} = arctan2\left(-sin(J5_{1,2})R_4^6[2,3], -sin(J5_{1,2})R_4^6[1,3]\right). \tag{4.30}$$

V jiných případech pokud $R_4^6[3,3] = 1$

$$J5_{1,2} = 0, \tag{4.31}$$

$$J6_1 = 0, J6_2 = \pi, \tag{4.32}$$

$$J4_1 = arctan2\left(-R_4^6[2,2], -R_4^6[1,2]\right), \tag{4.33}$$

$$J4_2 = arctan2\left(-R_4^6[2,2], -R_4^6[1,2]\right) + \pi \tag{4.34}$$

a pokud $R_4^6[3,3] = -1$

$$J5_1 = \pi, J5_2 = -\pi, \tag{4.35}$$

$$J6_1 = 0, J6_2 = \pi, \tag{4.36}$$

$$J4_1 = arctan2\left(R_4^6[2,2], R_4^6[1,2]\right), \tag{4.37}$$

$$J4_2 = arctan2\left(R_4^6[2,2], R_4^6[1,2]\right) + \pi. \tag{4.38}$$

Dále, podle rozsahu kloubových souřadnic, jsou odebrána řešení, která tyto podmínky nesplňují a přidána periodická řešení, pokud to klouby umožňují.

### 4.2.3 Diskuze počtu řešení

Pro některé polohy může nabývat manipulátor až osmi řešení, pokud uvažujeme pouze úhly v rozsahu $\langle -\pi, \pi \rangle$. Pro některé speciální polohy však může nabývat nekonečného počtu řešení. Tento případ nastává ve chvíli, kdy jsou alespoň dvě osy kloubů $J1$, $J4$ a $J6$ totožné.

# Kapitola 5

# Implementace řídícího systému v prostředí ROS

V této kapitole se budu zabývat vytvořením vizualizačního a kolizního modelu. Dále uvedu možnostmi začleněním vlastního řešení inverzní kinematické úlohy a nakonec vytvoření rozhraní mezi počítačem a řízeným hardwarem.
Pro vytvoření řídícího rozhraní byl vybrán systém ROS distribuce Melodic. Tuto verzi jsem zvolil, protože se v době zahájení mé práce jednalo o nejnovější stabilní verzi systému ROS.
Při vytváření tohoto systému jsem vyšel z implementace uvedené v projektu [16].

| Balíček | Zajištěné funkce |
|---|---|
| `mitsubishi_arm_config` | Nastavení mezních kloubových hodnot, plánování a řešení kinematických úloh |
| `mitsubishi_arm_control` | Konfigurace kontrolérů |
| `mitsubishi_arm_description` | Vizualizační a kolizní modely robotů |
| `mitsubishi_arm_kinematics` | Integrace inverzní kinematické úlohy do systému ROS |
| `mitsubishi_arm_launch` | Spuštění potřebných `.launch` souborů pro daný robot |
| `mitsubishi_arm_move` | Nevyužit, ponechán z licenčních důvodů |
| `mitsubishi_arm_hardware_interface` | Ovládací rozhraní, kódy pro řídící jednotky |
| `mitsubishi_arm_student_interface` | Modul pro snadné ovládání robotů |
| `schunk_grippers` | Funkce chapadla PG 70, Vizualizační a kolizní model |
| `kinematics_6DOF` | Řešení kinematických úloh robotů s 6DOF |

**Tabulka 5.1:** Vytvořené balíčky a jejich účel

Tabulka 5.1 obsahuje seznam použitých balíčků, jejichž funkce bude dále popsána. Pro bližší vhled do problematiky a pro konkrétní syntaxi je možné nahlédnout do přiložených zdrojových kódů na CD. Návod k instalaci, nastavení komunikačních rozhraní a použití systému je také k nahlédnutí v příloze na CD. Tato dokumentace je pro snazší šiřitelnost psána v angličtině.

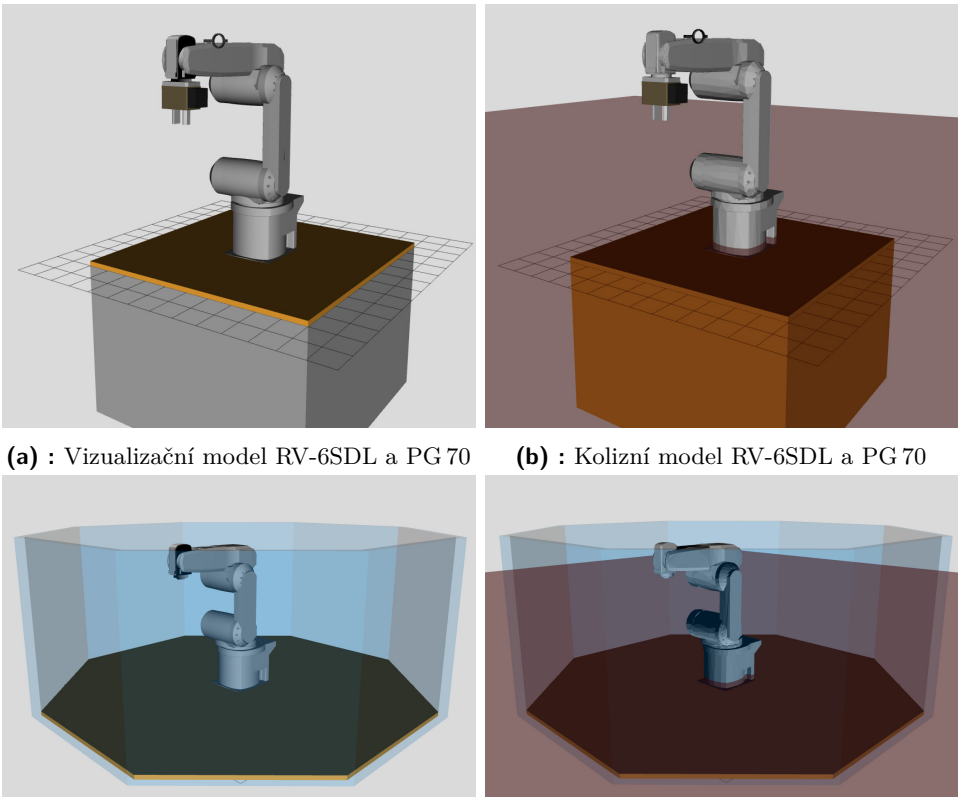
## 5.1 Kolizní a vizualizační model

V robotických aplikacích je často nutné předcházet jakýmkoliv kolizím robotu ať už se sebou samým a nebo s okolím. Výrobci manipulátorů tento problém řeší omezením pracovního prostoru v řídících jednotkách. Použité roboty Mitsubishi mají možnost tuto oblast definovat jako kvádr, případně jako jiný mnohostěn v prostoru. Takové řešení ovšem omezuje pouze možnosti pohybu koncového bodu manipulátoru. Neřeší pozici zbytku robotu, jeho ramen a v případě použití chapadla ani přenášeného objektu. Do průmyslových aplikací, kde je kladen důraz spíše na jednoduchost a spolehlivost, je toto řešení vhodné. Robot je obvykle uzavřen v ochranné kleci a jeho okolí je neměnné. V této aplikaci je sice také robot uzavřen za bezpečnostní barierou, ale s každou úlohou se mění jeho okolí. Dále je důležité si uvědomit, že s robotem budou pracovat nezkušení studenti a je potřeba zajistit co největší míru bezpečnosti.
Kolize jsou v systému ROS řešeny pomocí vizualizačního a kolizního modelu manipulátoru. Aby bylo možné tuto vlastnost systému využít, musí být takový kolizní model vytvořen. Současně s ním vznikne také vizualizační model, který je obvykle z grafického hlediska detailnější.
Modely jsou vytvořeny pomocí souboru ve formátu `.urdf` (Universal Robot Description File). Tento typ souboru je založen na `.xml` a umožňuje modelovat robot nejen pomocí geometrických útvarů jako jsou kvádry, koule a válce, ale také pomocí grafických 3D modelů ve formátu `.stl`.
Jednotlivá ramena jsou spojována klouby, tak jako u fyzického robotu. Formát `.urdf` poskytuje možnost využít klouby rotační a posuvné. Součástí definice kloubu je určení jeho mezních hodnot, případně i rychlosti a momentu. Parametry kloubů je možné upravit pomocí souboru `joint_limits_X.yaml` v balíčku `mitsubishi_arm_config` beze změny modelu, kde `X` představuje variantu robotu. Modely v tomto projektu mají klouby omezené pouze mezními hodnotami manipulátoru a bezpečnostní rezervou. Robot je pak omezen podle potřeby konfiguračním souborem typu `.yaml`.
Společnost Mitsubishi poskytuje grafické 3D modely svých robotů v souborech typu `.step`. Model robotu RV-6SDL byl již do jisté míry hotový v repositáři projeku [16] a tak mi zbývalo vytvořit model pro robot RV-6S. Soubor typu `.step` jsem pomocí programu FreeCAD rozdělil na dílčí součásti a exportoval je do souborů typu `.stl`. Obdobným způsobem, jakým byl tvořen model RV-6SDL, jsem vymodeloval i robot RV-6S.
Formát `.urdf` umožňuje snadné vytvoření kolizního modelu robotu. K definici každého ramene se přidá těleso, pro které je detekována kolize. Tento grafický objekt by neměl být příliš detailní, aby byla zachována dostatečná

rychlost vyhodnocení. V některých případech je kolizní model tvořen pouze jednoduchými geometrickými tělesy, která jsem zmínil výše. V projektu [16] byla využita možnost použít grafické soubory jako pro vizualizační model a snížit jejich komplexitu. Výsledný kolizní model je přesnější než model tvořený geometrickými tělesy a neomezuje příliš pohyb robotu [1].
Z modelu robotu RV-6SDL publikovaného v projektu [16] bylo třeba odebrat těleso představující momentový senzor. Dále jsem opravil drobnou nepřesnost, kdy šestý rotační kloub J6 byl umístěn mezi odstraněným momentovým senzorem a posledním ramenem robotu. Správně má být umístěn mezi šestým a sedmým ramenem. Tato nepřesnost nemá vliv na výslednou kinematiku, ale znemožňuje používání robotu bez připevněného nástroje.
Aby se ulehčilo vytváření modelů robotů v různých situacích a s různými nástroji, jsou modely robotů importovány do souborů, které popisují modely robotů již na konkrétních pracovištích. Tyto soubory jsou obsaženy v balíčku `mitsubishi_arm_description` ve složce `robots`.
Pro oba roboty jsem vytvořil modely jejich pracovišť se stoly a ochrannými klecemi. Pro robot RV-6SDL doposud nebyla smontována klec. Počítá se s její kompletací až při jeho trvalém umístění v učebně. Proto ani doposud používaný model robotu RV-6SDL klec neobsahuje[2]. K oboum robotům jsem dále přidal těleso s názvem `planning_tip`. Toto těleso nemá žádnou grafickou podobu a slouží pouze jako koncový bod robotu pro plánování.
Pro zvýšení bezpečnosti jsou v řídících jednotkách omezeny pracovní prostory manipulátorů na kvádry o rozměrech $20000 \times 20000 \times 9950\ mm$. Aby byla i tato skutečnost zohledněna v kolizním modelu daného robotu, byla přidána plocha `limitnig_plain` do výšky 50 $mm$ nad patu manipulátoru. Teoreticky by měly být přidány i ostatní stěny kvádru, ale na ty robot nemůže dosáhnout a tak jsou vynechány.
Na závěr byl k modelu robotu RV-6SDL připojen model chapadla Schunk PG 70, jehož model byl z velké části převzat z projektu [17]. Obrázky 5.1 zachycují finální podoby modelů.

[1] Jednoduchá tělesa by nekopírovala povrch ramene robotu a tam, kam by se robot ještě vešel, by již mohly působit detekci kolize.

[2] Model obsahuje těleso klece bez jakékoliv vizuální či kolizní podoby.



**(a) :** Vizualizační model RV-6SDL a PG 70

**(b) :** Kolizní model RV-6SDL a PG 70

**(c) :** Vizualizační model RV-6S

**(d) :** Kolizní model RV-6S

**Obrázek 5.1:** Výsledné modely

## 5.2 Implementace řešení kinematických úloh

Byl vytvořen soubor s názvem `kinematics_6DOF.py`.V tomto souboru jsem implementoval třídu `KIN_6DOF`, která obsahuje metody provádějící výpočty popsané v kapitole 4. Modul využívá knihovny `numpy` a `tf.transformations`. Při inicializaci třída vyžaduje délky ramen, extrémní hodnoty kloubů a transformaci z koncového systému robotu do systému nástroje.
Inverzní kinematickou úlohu řeší metoda `ikt(self, X)`, kde bod `X` je požadovaný poziční vektor koncového systému ve formátu `numpy.array` podle 2.7. Funkce navrací list listů všech řešení úlohy[3].
Klouby, pro které existuje nekonečný počet řešení, jsou označeny písmenem `'i'`. Výsledek pak může mít tvar například `[[0, 0, 0, 'i', 0, 'i']]`. Třída `KIN_6DOF` poskytuje metodu `ik_for_inf(self, X, J)`, která pro požadovanou polohu a orientaci koncového efektoru a dodefinované kloubové souřadnice dopočítá, poslední kloubovou souřadnici. Dodefinování musí být takové, aby zbyla pouze jedna neznámá hodnota. Uživatel si tak sám volí jakou konfiguraci robotu požaduje.
Dále třída `KIN_6DOF` umožňuje řešení přímé kinematické úlohy pro jakýkoliv systém kloubu a systém koncového efektoru. K tomuto účelu slouží funkce

[3]Matici, kde každý řádek představuje jedno řešení

`dkt_to_system(self, J, n)`, kde `J` představuje list kloubových souřadnic. Přirozené číslo `n` specifikuje, pro který systém je přímá kinematická úloha řešena. Řešení přímé kinematické úlohy čistě pro koncový systém navrací metoda `dkt(self, J)`, kde `J` je list kloubových souřadnic. Obě tyto metody navrací polohový vektor typu `numpy.array` podle 2.7.
Třída `KIN_6DOF` obsahuje další pomocné funkce, které nejsou primárně určeny k použití uživatelem. Pro bližší informace je možné nahlédnout do přiloženého zdrojového kódu a dokumentace na CD.

## 5.3 Začlenění řešení inverzní kinematické úlohy

V této části práce budu přistupovat k výpočtu inverzní kinematické úlohy jako k již hotové funkci popsané v kapitolách 5.2 a 4.2. Tuto funkci pro zjednodušení označím $F$.
Jakýkoliv kinematický plugin[4] začleněný do systému ROS musí být třídou odvozenou od třídy `kinematic_base` v jazyce C/C++. Stejného typu je také základní kinematický plugin balíčku MoveIt, `KDLKinematicsPlugin`.
Jelikož psaní vlastního kinematického pluginu může být poměrně náročné, existuje možnost využít `SRVKinematicsPlugin`. Tento kinematický plugin, jak napovídá název, při řešení inverzní kinematiky vznese žádost na server poskytující službu `/solve_ik`. [11]
Přepínání mezi kinematickými pluginy pro plánovací skupinu `arm` je prováděno v souboru `kinematics.yaml` v balíčku `mitsubishi_arm_config`. Zde je také možné nastavit další parametry, jako je maximální počet pokusů pro vyřešení inverzní kinematické úlohy, maximální čas a požadovaná přesnost výsledku. Mnou vytvořený analytický kinematický plugin tyto dodatečné parametry ignoruje.
Jelikož jsem měl již hotový algoritmus řešení inverzní kinematické úlohy připravený v jazyce Python, rozhodl jsem se pro variantu s využitím `SRVKinematicsPlugin`. Nevýhodou zůstává, že tato varianta v současné době stále nemá implementovanou podporu výpočtů přímé kinematické úlohy. [11] Tato skutečnost ničemu nevadí, jelikož se k řešení přímé kinematické úlohy v systému nevyužívá[5].
Vytvořil jsem skript v jazyce Python `mitsubishi_kinematics_server.py` v balíčku `mitsubishi_arm_kinematics`. Obsahem tohoto skriptu je vytvoření serveru poskytujícího službu `/solve_ik`.
Parametry pro výpočet jsou nahrány na parameter server před spuštěním skriptu pomocí souboru `mitsubishi_kinematics.launch`. Pouhou editací těchto údajů lze využít kinematický plugin i pro jiný robot. Takový manipulátor musí mít 6DOF, dostatečně podobnou geometrii prvních tří kloubů a osy jeho druhých tří kloubů se musí protnout v jednom bodě.
Struktura služby pro výpočet inverzní kinematiky je dána vzorem `GetPositionIK` a je k nahlédnutí na stránkách [12]. Při požadavku na výpočet inverzní kinematiky je nejprve převedena reprezentace rotace z kvaternionů

[4] Program, který řeší kinematické úlohy.
[5] Není volán kinematický plugin.

do Eulerových úhlů, pomocí knihovny `tf.transformations`. Přepočtený polohový vektor je předán funkci $F$.
Nalezne-li $F$ více validních řešení, je vybráno takové, které splňuje

$$\min_{i \in n} \left( |\boldsymbol{J_s} - \boldsymbol{J_i}| \right), \tag{5.1}$$

kde $n$ je počet nalezených řešení a $J_s$ je při volání serveru dodaný stav robotu v kloubových souřadnicích.
Pokud některý výsledek obsahuje označení nekonečného počtu řešení podle konvence zavedené v kapitole 5.2, je nejprve získáno dodefinované řešení. Kloubové souřadnice jsou dodefinovány pomocí dodaného stavu robotu tak, aby zbyla jedna neznámá kloubová hodnota. Dodefinované klouby jsou voleny tak, aby ne-dodefinovaný kloub měl nejvyšší pořadové číslo. Voláním příslušné funkce je získáno konkrétní řešení pro dodefinované kloubové souřadnice. Následně je vybráno řešení podle kritéria 5.1.
S nalezeným řešením je navrácen chybový kód `MoveItErrorCodes.SUCCESS`. Pokud funkce $F$ nenalezne řešení, je navrácený chybový kód `MoveItErrorCodes.NO_IK_SOLUTION`. I tak je nutné navrátit kloubové souřadnice, ačkoli na hodnotách nezáleží.

## 5.4 Rozhraní mezi počítačem a robotem

### 5.4.1 Ze strany robotu

Bylo potřeba zjistit, jaké možnosti komunikace roboty nabízejí. Jak již bylo zmíněno v kapitole 3.1, je robot RV-6SDL, přesněji jeho řídící jednotka, vybaven Ethernetem 100Base-T a sériovou linkou RS-232. Druhý robot, RV-6S, disponuje pouze rozhraním RS-232.
Protokol použitý pro řízení robotu RV-6SDL přes rozhraní Ethernet se nazývá `MXT Control`. Umožňuje posílat robotu souřadnice, ať už kartézské či kloubové, v časových intervalech minimálně 7 $ms$.[4] Pokud je zaslána nová souřadnice před dokončením pohybu do dříve zaslané pozice, robot okamžitě zahájí pohyb ze současné pozice do nově zadané pozice. Pomocí `MXT protokolu` je možné zjistit současnou pozici koncového bodu robotu, či kloubové souřadnice. Pro otevření takového ovládacího kanálu je třeba v programu jednotky volat příkaz `Mxt 1,1,100`. Pro ukončení vzdáleného ovládání je třeba zaslat k tomu určený typ datového packetu.[4] Použitý zdrojový kód se jmenuje `1.txt` a je k nahlédnutí na přiloženém CD.
Při ovládání robotu pomocí RS-232 je použit Melfa protokol. Tento protokol byl, pravděpodobně, navržen hlavně na diagnostiku a správu řídící jednotky. Umožňuje zadat pohyb robotu, není ovšem možné tento pohyb přerušit nebo změnit požadovanou pozici robotu tak snadno jako u protokolu `MXT control`. Proto jsem vytvořil program v jazyce MELFFABSIC IV, který je možné popsat následovně.
Jedná se o jednoduchou smyčku typu `while`, která je vykonávána dokud je proměnná `M1` rovna jedné. V těle této smyčky robot nastavuje pozici danou proměnnou `J1` v kloubových souřadnicích. `Melfa protokol` umožňuje

příkazem `HOT` měnit za chodu programu jeho proměnné. Změnou proměnné `J1` je nastavována požadovaná pozice koncového bodu robotu. Stejně tak změnou proměnné `M1` program opustí smyčku a ukončí se. Celý zdrojový kód je k nahlédnutí na přiloženém CD v souboru `Moveit.txt`. Čtení současných kloubových hodnot je prováděno také pomocí `Melfa protokolu` příkazem `JPOSF`.

### 5.4.2 Ze strany systému ROS

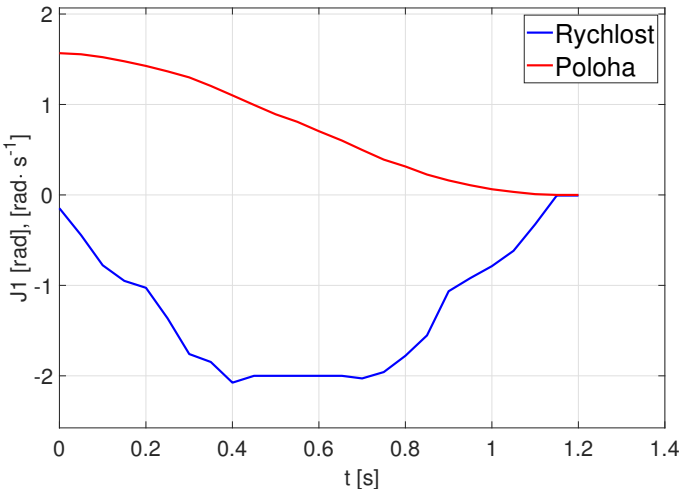
Další postup obnáší začlenění ovládacího protokolu manipulátoru do systému ROS a MoveIt. K tomu byl využit balíček ROS control. Tento balíček nabízí rozhraní pro čtení kloubových souřadnic robotu a pro jejich zapisování. Je možné si vybrat z pozičních, rychlostních a momentových ovladačů. Nabízí také ovladač pro řízení trajektorie. Byl zvolen `Joint Trajectory Controller`, který podle dodaného plánu posílá prostřednictvím komunikačního rozhraní kloubové souřadnice řídící jednotce. Rychlost pohybu je řízena pomocí počtu bodů na jednotku vzdálenosti. Pro zavedení zpětné vazby je využito rozhraní `Joint State Interface`.[13]
Při použití balíčku ROS control je třeba dodat definice rozhraní pro klouby do modelu robotu typu `.urdf`. Jedná se o označení kloubů, pro které je vytvořeno rozhraní.
Pro správnou funkci je třeba vytvořit třídu v jazyce C/C++ obsahující metody `readHW()` a `writeHW()`, které zajišťují komunikaci s robotem. Informace o stavu robotu a požadovaných akčních zásazích se mezi metodami a kontrolérem předávají pomocí funkcionality `Joint State Handle` balíčku Hardware Interface.
Při inicializaci rozhraní je vytvořen objekt `ControllerManager`, který vytvoří akční server a propojí rozhraní se systémem ROS a MoveIt. Dále je navázáno spojení s robotem a pomocí `Melfa protokolu` je spuštěn příslušný program v řídící jednotce.
Pokud je požadována pouze simulace bez použití fyzického robotu, je použit stejný princip. Pouze při volání funkce `writeHW()` jsou předávány hodnoty kloubů přímo do zpětné vazby.
Aby systém ROS věděl, jaký regulátor má být použit, je v souborech `mitsubishi_arm_control.yaml` provedena jeho konfigurace. Je zde specifikováno, jaké klouby jsou řízeny, jaká je požadována frekvence zápisu a čtení. Případně jsou nastaveny další parametry řízení jako například maximální odchylka požadované a vykonané trajektorie[14]. Konkrétní nastavení jsou uvedena v přiloženém zdrojovém kódu na CD.
Zdrojový kód pro robot RV-6SDL je z velké části převzat z projektu [16]. Pro robot RV-6S je tento kód upraven, aby podporoval jiné komunikační rozhraní.

### 5.4.3 Plánování trajektorie

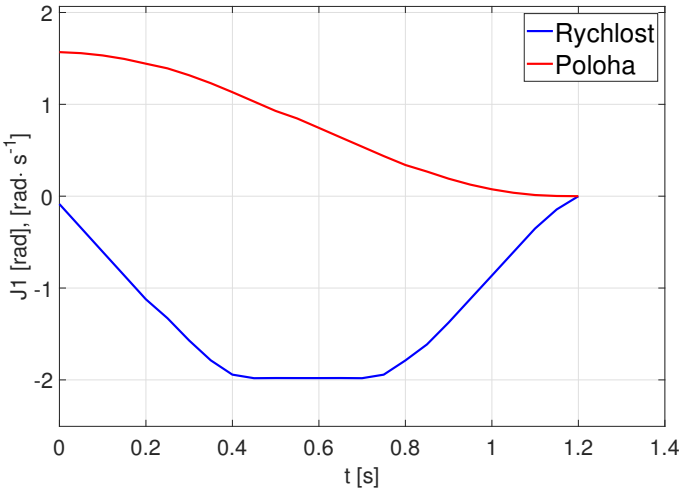
Balíček MoveIt používá pro plánování pohybu knihovnu OMPL, Open Motion Planning Library. Pro správné naplánování je potřeba dodat informace o limitních vlastnostech kloubů, jako je rozsah, rychlost a zrychlení. Tyto

hodnoty jsou získávány ze souboru typu `joint_limits.yaml`, nebo případně přímo z modelu robotu ve formátu `.urdf`, není-li soubor typu `.yaml` dodán. Knihovna OMPL nabízí tři algoritmy časové parametrizace, Iterative Parabolic Time Parameterization, Iterative Spline Parameterization a Time-optimal Trajectory Generation. Postupně jsem algoritmy vyzkoušel a vybral ten, který poskytoval nejlepší výsledky z pohledu plynulosti pohybu. Na Obrázku 5.2 jsou zachyceny trajektorie naplánované různými algoritmy. Pro jednoduchost uvádím pouze plán pro kloub J1 a bez skutečného provedení.

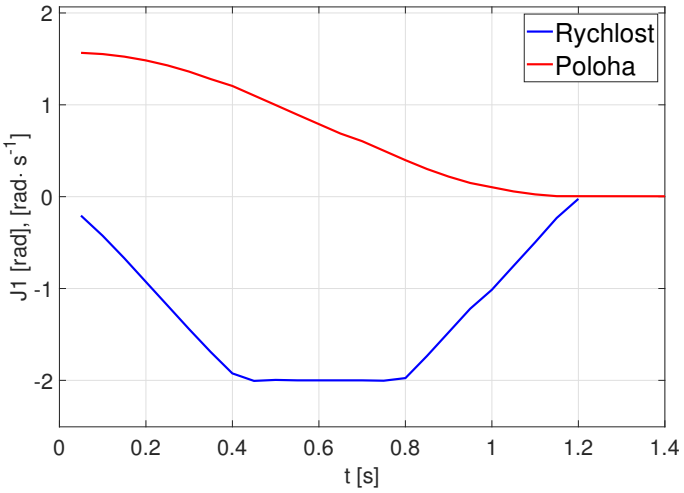
Algoritmus Iterative Parabolic Time Parameterization při plánování překročil maximální povolenou rychlost robotu. Na webových stránkách balíčku MoveIt se uvádí, že tento algoritmus obsahuje chybu[10]. Dále je průběh rychlosti značně zubatý. Proto není vhodný pro využití v rozhraní.
Druhé dva algoritmy navrhly velmi podobné trajektorie. Průběh rychlosti od algoritmu Iterative Spline Parameterization je plynulejší. Dále byl algoritmus Time-optimal Trajectory Generation zařazen do systému později než Iterative Spline Parameterization[10]. Z těchto důvodů jsem se rozhodl pro využití algoritmu Iterative Spline Parameterization.




**(a) :** Iterative Parabolic Time Parameterization




**(b) :** Iterative Spline Parameterization

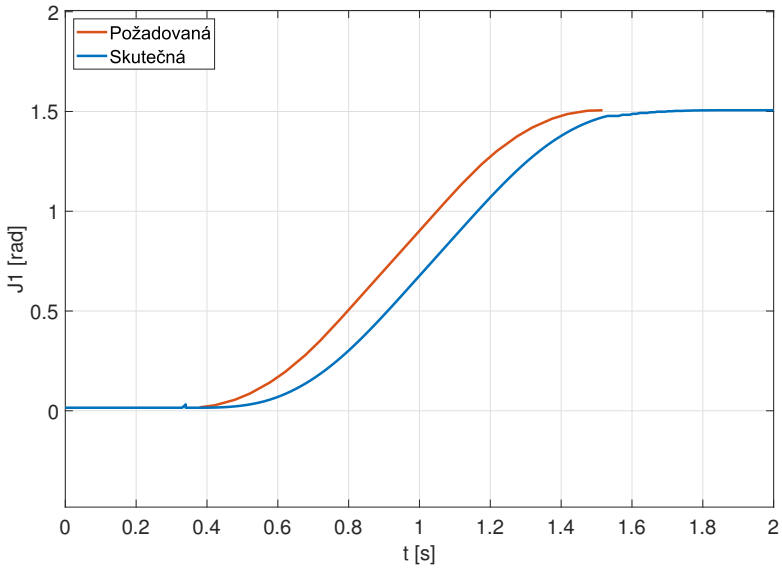



**(c) :** Time-optimal Trajectory Generation

**Obrázek 5.2:** Srovnání algoritmů časové parametrizace

### 5.4.4 Konfigurace plánování a reálný pohyb robotů

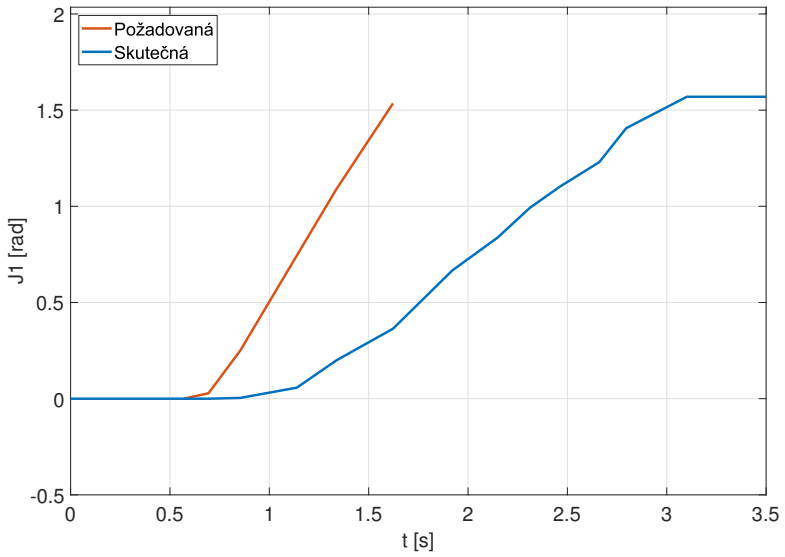
Maximální kloubovou rychlost pro jednotlivé klouby bylo potřeba pro robot RV-6SDL experimentálně určit. V případě využití `MXT control` si jednotka, pravděpodobně, z požadované pozice dopočítává, jakou rychlostí by měla klouby pohybovat, aby překonala zadanou vzdálenost v daném čase. Pokud řídící jednotka dojde k závěru, že nemůže stihnout do požadované pozice dorazit v čas, přejde do chybového stavu, celý robot se zastaví a je třeba celý systém restartovat. Bohužel se výrobce nezmiňuje, o jak dlouhý časový úsek se jedná. Výrobce v dokumentaci neuvádí maximální zrychlení kloubů robotu [3]. Tyto parametry byly v prostředí ROS postupně navyšovány, dokud řídící jednotka takovýto pohyb umožnila.




**Obrázek 5.3:** Požadovaná a uskutečněná trajektorie pro robot RV-6SDL

Průběh regulace na Obrázku 5.3 má vhodný průběh pro řízení trajektorie. Při komunikaci nedochází k výraznému dopravnímu zpoždení.
Stejným způsobem byl proveden odhad rychlostí a zrychlení i pro robot RV-6S. Zásadním omezením se projevila rychlost rozhraní RS-232. Ačkoliv dokumentace řídící jednotky uvádí maximální přenosovou rychlost 19200 *bps*, je možné nastavit rychlost až 38400 *bps*. Příkaz `Melfa protolu` je tvořen textovým řetězcem a jako takový je neefektivní z hlediska rychlosti přenosu. Při řízení není přenosová rychlost dostatečně vysoká, aby doba zjištění současné polohy a zadání nové pozice robotu byla vůči době vykonávání dříve zadaného pohybu zanedbatelná. Při snížení maximální dovolené rychlosti robotu[6] je pohyb přijatelně plynulý, viz Obrázek 5.4.

[6] A tím zvýšení doby vykonávání pohybu.




**Obrázek 5.4:** Požadovaná a skutečněná trajektorie pro robot RV-6S

Na Obrázku 5.4 je vidět značná regulační odchylka řízení trajektorie. Taková regulace by mohla být v některých aplikacích nevhodná. Při použití více robotů současně by byla nepřijatelná. Pro účely této práce je však dostatečná. Za účelem zlepšení regulace jsem zvažoval vytvoření komplexnějšího programu pro řídící jednotku. Tento program by využíval multitasking [7]. Hlavní program by obsluhoval regulaci polohy podobným způsobem jako současný algoritmus. Podprogram by zajišťoval komunikaci pomocí RS-232 a nastavoval by požadovanou polohu pro hlavní program. Data by byla odesílána bez jakéhokoliv protokolu, pouze jako šestice čísel. Pokud by byl jeden číselný údaj kódován pomocí formátu IEEE754, bylo by možné dosáhnout následující maximální frekvence záspisu a čtení.

$$B = 34800\ bps, F = 12b, T = 6 \cdot 4 \cdot F,$$
$$f_{max} = \frac{B}{2T} \doteq 66.67\ Hz$$

Parametr $F$ představuje šířku jednoho rámce RS-232 (jeden start bit, 8 datových bitů, jeden paritní a 2 stop bity).
Pro srovnání, současná maximální frekvence zápisu a čtení je určena jako

$$N_1 = 68, N_2 = 10, N_3 = 100$$
$$f_{max} = \frac{B}{F\,(N_1 + N_2 + N_3)} \doteq 17.98\ Hz,$$

kde $N_1$, $N_2$ a $N_3$ představují počet přenášených znaků v nejhorším případě. Při výpočtu jsem uvažoval následující textové řetězce:

1;1;HOTMoveit;J1=(-100.00,-100.00,-100.00,-100.00,-100.00,-100.00,)\r

[7] Funkcionalita jazyku MELFABASIC, kdy je využit hlavní program a podprogramy.

1;1;JPOSF\r
QoKJ1;-100.00;J2;-100.00;J3;-100.00;J4;-100.00;J5;-100.00;J6;-100.00;L1;0.00;;6,0;100;0.00;00000000\r

Bohužel se ukázalo, že jednotky robotů Mitsubishi neumožňují odesílat data jinak, než ve formátu ASCII. Čísla jsou odesílána po číslicích jako textový řetězec a není možné je odesílat jinak.[5]
`Melfa protokol` nebyl, podle všeho, navržen pro vzdálené řízení trajektorie v reálném čase. Druhou možností řízení robotu je použít poziční regulátor. Mnou zvolený postup umožňuje regulaci rychlosti vykonávaného pohybu. Pokud by se v budoucnu ukázalo toto řešení jako nevhodné, lze jej nahradit pozičním regulátorem.

## 5.5 Rozhraní mezi počítačem a Schunk PG 70

### 5.5.1 Implementace rozhraní

Jak již bylo zmíněno, při implementaci chapadla Schunk PG 70 v systému ROS jsem využil projekt [17]. Tento projekt realizoval rozhraní pomocí služeb, které umožňovaly s chapadlem manipulovat a zajišťovat provozní záležitosti jako je třeba potvrzování chyb [8]. Tento systém je funkční, ale já jsem se rozhodl využít balíček ROS control a akční server typu `Gripper Command`. Toto řešení umožňuje pohybovat robotickou rukou a chapadlem současně.
Jelikož pro tento typ akčního serveru není vytvořeno v systému ROS rozhraní jako pro typ `Joint Trajectory Controller`[9], vytvořil jsem v jazyce Python skript, který ho nahrazuje.
Pro začlenění rozhraní do systému ROS jsem vytvořil uzel s funkcí akčního serveru. Tento uzel jsem pojmenoval `pg70_control`. Jednou z vlastností uzlu poskytujícího službu v prostředí ROS je, že poskytnutím služby klientu není přerušena činnost uzlu, který službu poskytuje. Uzel `pg70_control` zajišťuje komunikaci s chapadlem.
Komunikace s chapadlem je implementována obdobně jako rozhraní pro robotické paže. Byly vytvořeny dvě funkce `readHW(self)` a `writeHW(self)`, které jsou volány uzlem `pg70_control` s frekvencí 100 $Hz$. Tyto funkce náleží třídě `pg70_hw_interface`.
Voláním akčního serveru je spuštěna funkce `execute_cb(self,goal)` třídy `pg70_hw_interface`. Parametr `goal` obsahuje novou požadovanou polohu prstů chapadla. Nastavením příslušného příznaku je vytvořen příkaz k odeslání zprávy chapadlu. Při následujícím volání funkce `writeHW(self)` je zpráva s příkazem ke změně polohy odeslána.
Metoda `execute_cb(self, goal)` dále čeká dokud není dosažena požadovaná pozice prstů chapadla a přitom kontroluje, zda nebyl přesažen čas stanovený jako maximální dovolený pro vykonání úkonu.

[8] Error acknowledgment.
[9] Já jsem je alespoň nenašel.

Pokud výkon akce proběhne v pořádku, je procesu, který server volal, navrácen stav `succeeded`. V opačném případě je nastavením příznaku vznesen požadavek na zastavení pohybu chapadla a je vrácen stav `aborted`. Příkaz opět chapadlu odesílá funkce `writeHW(self)`.
Metoda `readHW(self)` zajišťuje informace o současné poloze prstů chapadla a tyto hodnoty publikuje do topiku `/joint_states`.
Během provozu chapadla může dojít k chybám jako je například pokles napájecího napětí nebo překročení maximálního proudu vinutím motoru. Některé tyto chyby musí být potvrzeny, aby chapadlo dále vykonávalo svou činnost a reagovalo na zprávy.[8] Při čtení pozice prstů chapadla jsou vyčtena také chybová hlášení, jsou odsouhlasena a uživatel je o nich informován.

### 5.5.2 Maximální proud vinutím motoru

Z důvodu bezpečnosti jsem prováděl pokusy s maximálním proudem vinutím motoru. Tímto parametrem je možné nastavit maximální moment, kterým může motor chapadla působit. Silové účinky jsem pozoroval pomocí deformace molitanu a polystyrenu.
Při nastavení proudu pod 250 $mA$ byla deformace molitanu velmi nízká, zato se někdy nepodařilo motoru ani rozpohybovat převodovku chapadla. Pro rozpohybování vnitřního mechanismu byla, pravděpodobně, třeba větší síla, než jakou byl takto omezený motor schopen působit.
Bylo třeba nalézt kompromis mezi spolehlivostí a bezpečností aplikace. Proto jsem omezil proud na 850 $mA$. Při tomto proudu už motor měl možnost dodat dostatečný moment pro rozpohybování mechanismu. Na druhou stranu je možná působená deformace vyšší. Pro větší flexibilitu systému jsem přidal možnost nastavení maximálního proudu do souboru typu `.launch` spouštějícího rozhraní.

# Kapitola 6

# Rozhraní pro studenty

Plné pochopení systému ROS není úplně snadné. V rámci výuky robotiky na ČVUT se neprobírá a bylo by pro studenty obtížné roboty ovládat. Proto byl vytvořen modul v jazyce Python 2.7, který zapojení vlastního uživatelského kódu usnadňuje a předkládá studentům již hotový nástroj. Modul obsahuje třídu `Mitsubishi_Robot`.
Z části tato třída zapouzdřuje funkce dodávané modulem `MoveItCommander`. Přidává však další vlastní funkcionality, jako je řešení kinematických úloh a zadávání výkonu trajektorie pomocí kloubových souřadnic. Dále omezuje rychlost vykonávání pohybu robotu RV-6SDL na 30 % maximální rychlosti. Robot RV-6S je ponechán s plnou rychlostí pohybu. Toto rozhodnutí je opodstatněné negativními vlastnostmi jeho rozhraní popsaných v kapitole 5.4.4 a jeho celkovým zpomalením. Dokumentace funkcí modulu je na přiloženém CD.
Dalším nástrojem vytvořeným pro studenty je kinematická kalkulačka. Jedná se o jednoduchý skript v jazyce Python umožňující provádět výpočty přímé a inverzní kinematické úlohy. S uživatelem komunikuje pomocí dialogů. Tento skript je vhodný pro přípravu, kdy si studenti mohou před-počítat kloubové souřadnice pro zadané polohy.
K funkci tohoto programu není potřeba systém ROS. Rozhodl jsem se pro toto řešení, aby se studenti mohli s jeho pomocí připravovat doma a nemuseli si ROS instalovat. Zdrojový kód kalkulačky je k nahlédnutí v příloze na CD.

# Kapitola 7
# Praktická úloha

Jako finální test funkčnosti celého systému řízení robotu, vizualizace a plánování jsem vypracoval praktickou úlohu. Podobná úloha by mohla být součástí praktických cvičení při výuce robotiky.

## 7.1 Zadání

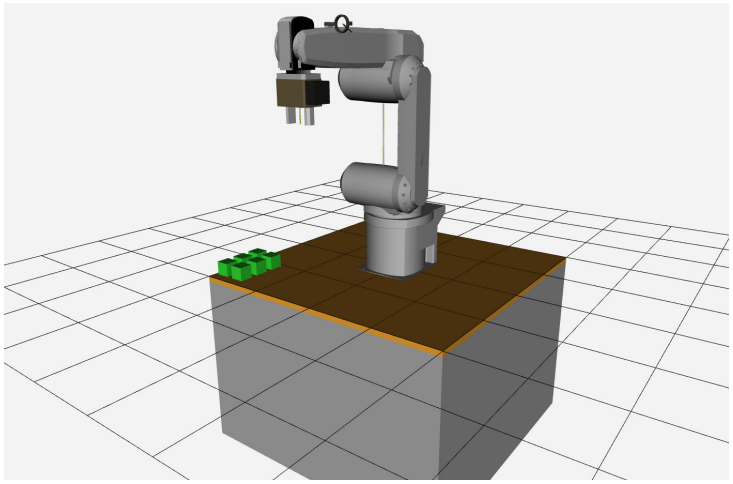
Z dodaných kostek tvaru krychle o délce hrany 50 *mm* postavte trojúhelníkovou zeď. Stavbu bude tvořit šest kostek podle Obrázku 7.1. Využijte robot Mitsubishi RV-6SDL s chapadlem Schunk PG 70. Kostky jsou na začátku úlohy uloženy na okraji pracovního prostoru a jejich umístění je známé. Kostky budou položeny před robotem na pracovní desce. V této části je pracovní prostor robotu spojitý.

**Obrázek 7.1:** Požadovaná stavba

## 7.2 Postup

Byl vytvořen skript v jazyce Python 2.7. Nejprve proběhne vytvoření objektu `robot` třídy `Mitsubishi_Robot` a inicializace pracoviště. Inicializace spočívá ve vytvoření krychlí ve vizualizačním prostředí Rviz. Robot je následně pomocí nastavení kloubových souřadnic uveden do základní pozice

$$J_0 = \left[0, 0, \tfrac{\pi}{2}, 0, \tfrac{\pi}{2}, 0\right]^T. \tag{7.1}$$



**Obrázek 7.2:** Stav robotu a pracoviště před začátkem práce

Při samotném stavění robot nejprve najede do bodu 100 $mm$ nad kostku, poté ji uchopí a vrátí se opět do bodu 100 $mm$ nad místo, kde stála kostka. Poté může robot kostku bezpečně odnést nad místo uložení. Při tomto pohybu otočí chapadlo o úhel $\frac{\pi}{2}$ aby prsty chapadla nepřekážely při pokládání kostky do řady. Kostka je puštěna 50 $mm$ nad deskou stolu, případně nad jinou kostkou ve vyšších patrech stavby. Kostka sice při pokládání padá, ale snižuje se tím riziko kolize chapadla s okolím. Dále se koncový bod opět posune o 100 $mm$ nad místo uložení kostky s otočením chapadla o $-\frac{\pi}{2}$ se vydá pro další kostku. Po uložení poslední kostky se vrátí manipulátor do základní polohy 7.1.
Zdrojový kód řešení úlohy je dostupný na přiloženém CD.

# Kapitola 8

# Závěr

Na základě již existujících projektů jsem navrhl funkční robotická pracoviště s roboty RV-6SDL a RV-6S využívajících systém ROS. Při návrhu jsem kladl důraz na výukový charakter pracovišť. Vytvořil jsem modul pro jazyk Python usnadňující práci s robotickou buňkou.
Rozhraní pro ovládání robotu RV-6SDL funguje plynule bez poruch a umožňuje řízení trajektorie v reálném čase. Komunikační rozhraní s RS-232 se ukázalo být pro řízení trajektorie robotu RV-6S v plné rychlosti příliš pomalé. Po snížení maximální rychlosti manipulátoru bylo řízení trajektorie dostatečně plynulé. Toto řešení ovládacího rozhraní umožňuje regulovat rychlost pohybu. Pokud by se v budoucnu toto řešení ukázalo jako nevhodné, lze ho nahradit poziční regulací.
Robot RV-6SDL je vybaven chapadlem Schunk PG 70. Navržené rozhraní pro toto chapadlo je mimo základní funkci schopno reagovat na případná chybová hlášení. Z bezpečnostních důvodů je omezen maximální moment, jakým může chapadlo působit.
Vytvořil jsem analytické řešení inverzní kinematické úlohy pro oba roboty. Řešení jsem integroval do systému ROS pomocí odpovídající služby. Tento algoritmus jsem používal při testování manipulátorů a mohu potvrdit, že poskytuje správné výsledky.
Implementované řešení inverzní kinematické úlohy by mělo být funkční nejen pro roboty RV-6SDL a RV-6S, pro které bylo navrženo, ale po správné konfiguraci algoritmu i pro jiný robot. Takový robot musí mít šest stupňů volnosti, dostatečně podobnou geometrii prvních tří kloubů a osy posledních tří kloubů se musí protnout v jednom bodě.
Vytvořené vizualizační a kolizní modely zahrnují celá pracoviště včetně stolů, ochranných bariér a omezení zabudovaných v řídících jednotkách. Pro robot RV-6SDL nebyla bariéra dosud smontována. Aby model popisoval současný stav pracoviště, bariéru nahrazuje prázdné těleso. Použité kolizní modely zajišťují bezpečnost práce s manipulátory. Do projektu je možné zapracovat další varianty použitých robotů.
Pro pracoviště jsem vytvořil uživatelský manuál, popisující software i hardware projektu. Tento dokument zahrnuje návod k instalaci systému, správné nastavení komunikačních rozhraní a parametrů řídících jednotek. Dále manuál obsahuje návod k obsluze pracovišť a řešení možných chybových hlášení.

# Literatura


[1] JAZAR, Reza N. *Theory of applied robotics: kinematics, dynamics, and control.* 2nd ed. New York: Springer, c2010. ISBN isbn:978-1441917492.

[2] Mitsubishi: *RV-6S Series Standard Specifications Manual*, Japonsko, 2006

[3] Mitsubishi: *RV-6SD/6SDL Series Standard Specifications Manual*, 2012

[4] Mitsubishi: *CRnQ/CRnD Controller INSTRUCTION MANUAL - Detailed explanations of functions and operations*, Japonsko, 2009

[5] Mitsubishi: *CR1/CR2/CR3/CR4/CR7/CR8 Controller INSTRUCTION MANUAL - Detailed explanations of functions and operations*, Japonsko, 2003

[6] Mistubishi: *Manuál komunikačního porotokolu Mitsubishi*, dokument BFP-A4288-M, 2009

[7] Schunk GmbH & Co. KG.: *Assebly and operating manual PG*, ver. 11.00, 2018

[8] Schunk GmbH & Co. KG.: *Motion Control Schunk V 1.60*, ver. 1.00, 2015

[9] Martin Dubec: *Programové vybavení pro robotickou buňku*, diplomová práce ČVUT, 2011

[10] MoveIt: *MoveIt tutorials*, `http://docs.ros.org/en/melodic/api/moveit_tutorials/html/index.html`

[11] MoveIt: *Dokumentace pluginů MoveIt* `https://moveit.ros.org/documentation/plugins/`

[12] MoveIt: *Formáty zpráv a služeb MoveIt* `http://docs.ros.org/en/hydro/api/moveit_msgs/html/index-msg.html`

[13] Wim Meeussen: *Dokumentace ROS control*, `http://wiki.ros.org/ros_control`

[14] Adolfo Rodriguez Tsouroukdissian: *Dokumentace Joint Trajectory Controller*, `http://wiki.ros.org/joint_trajectory_controller`

[15] ROS: *ROS tutorials* `http://wiki.ros.org/ROS/Tutorials`

[16] The STORM Lab at Vanderbilt University: *mitsubishi_arm* `https://github.com/vustormlab/mitsubishi_arm`, verze: 6. 4. 2017

[17] Frantisek Durovsky: *schunk_grippers* `https://github.com/SmartRoboticSystems/schunk_grippers`, verze: 24. 9. 2015

# Příloha A

## Obsah přiloženého CD

Body následujícího seznamu jsou adresáře přílohy.

- **Dokumentace:**
  - **kinematics_6DOF.pdf** - Dokumentace kinematického python modulu
  - **mitsubishi_robots.pdf** - Dokumentace modulu rozhraní pro studenty
  - **Mitsubishi_workplace_documentation_and_manual.pdf** - Uživatelský manuál pracovišť
- **Fotografie pracovišť:**
  - Obsahuje fotografie robotických buněk.
- **Kinematická kalkulačka:**
  - **kinematic_calc.py** - Zdrojový kód kinematické kalkulačky
  - **kinematics_6DOF3.py** - Modul řešící kinematické úlohy, funkční bez ROS
  - konfigurační soubory v .txt
- **Praktická úloha:**
  - **brickmaster.py** - Zdrojový kód řešení praktické úlohy
- **Projekt:**
  - Zdrojové kódy a konfigurační soubory tvořící rozhraní v systému ROS. Blíže na CD

Items in following list are names of directories in appendix.

- Documentations:
  - **kinematics_6DOF.pdf** - Documentation of kinemtics python module
  - **mitsubishi_robots.pdf** - Documentation of student interface module
  - **Mitsubishi_workplace_documentation_and_manual.pdf** - User manual of workplaces
- Photos of workplaces:
  - Contains photos of robotic cells
- Kinematics calculator:
  - **kinematic_calc.py** - Source code of kinematics calculator
  - **kinematics_6DOF3.py** - Modul solving kinematics, functional without ROS
  - configuration files in .txt
- Practical task:
  - **brickmaster.py** - Source code of practical task solution
- Project:
  - Source codes and configuration files of ROS application. More on CD.