Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra kybernetiky

Bakalářská práce

Využití systému ROS pro plánování pohybových trajektorií robotů

Plzeň 2022 Tomáš Hefler

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta aplikovaných věd
Akademický rok: 2021/2022

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Tomáš HEFLER**
Osobní číslo:	**A19B0287P**
Studijní program:	**B0714A150005 Kybernetika a řídicí technika**
Specializace:	**Automatické řízení a robotika**
Téma práce:	Využití systému ROS pro plánování pohybových trajektorií robotů
Zadávající katedra:	**Katedra kybernetiky**

Zásady pro vypracování

- Seznamte se se systémem ROS a principu jeho použití
- Prozkoumejte možnosti modelování robotů a plánování jejich pohybu
- Soustřeďte se jak na standardní pohyby (přímkový, kruhový,...) tak i na komplexní pohybové trajektorie (např. interpolace/aproximace bodů, atd.)
- Sestavte model vybraného robotu a simulujte jeho pohyb po vybrané trajektorii
- Diskutujte možnosti přenosu naplánované pohybové trajektorie či dalších simulovaných dat do externího řídicího systému (REXYGEN)
- Diskutujte výsledky a další možnosti práce

Rozsah bakalářské práce: **30 – 40 stránek A4**
Rozsah grafických prací:
Forma zpracování bakalářské práce: **tištěná**

Seznam doporučené literatury:

W. Khalil, E. Dombre: Modelling, Identification and Control of Robots.
L. Sciavicco, B. Siciliano: Modelling and Control of Robot Manipulators.
Přednášky k předmětu URM.

Vedoucí bakalářské práce: **Ing. Martin Švejda, Ph.D.**
Výzkumný program 1

Datum zadání bakalářské práce: **15. října 2021**
Termín odevzdání bakalářské práce: **23. května 2022**

L.S.

Doc. Ing. Miloš Železný, Ph.D.
děkan

Prof. Ing. Josef Psutka, CSc.
vedoucí katedry

V Plzni dne 15. října 2021

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 20. května 2022

Tomáš Hefler

Abstract

Tato bakalářská práce se soustředí na možnosti modelování a řízení robotů pomocí systému ROS a plánování jejich pohybových trajektorií. Práce obsahuje informace o základních funkčních konceptech systému a možnostech jeho využití při práci s roboty. Popisuje potřebné programy pro tvorbu, vizualizaci a simulaci modelů a popis tvorby jednoduchého modelu. Součástí práce je také plánování jednoduchých i komplexnějších trajektorií pomocí programu MoveIt!. Dále pojednává o možnosti využití naplánovaných trajektorií v jiných řídících systémech. Ke konci práce jsou obsaženy informace o dalších možných aplikacích systému ROS a jeho částí.

Klíčová slova: ROS, MoveIt, rviz, plánování trajektorií

Abstrakt

This bachelor thesis focuses on potentialities of modeling and control of robots using the ROS system and planning of movement trajectories. Thesis concerns basic information about concepts of ROS and it's capabilities in robot development. It describes software needed for making, visualising a simulating of robot models and basic robot description making. Thesis also contains topic of simple and complex movement trajectories planning using the MoveIt! software. It also consists of possibility of exporting planned paths into other control systems. In the end of this thesis is information about other possible uses of the ROS system.

Key words: ROS, MoveIt, rviz, trajectories planning

Obsah

1 Úvod

V současné době probíhá po celém světě rozmach robotů, které dokáží vykonávat práce rychleji, přesněji a výhodněji než lidé. Ať už se jedná o průmyslový manipulátor, dron s autonomním řízením nebo v současnosti často zmiňované autonomní automobily, každý z těchto robotů musí obsahovat specifický řídící algoritmus, který určuje chování robotů a plánuje jejich budoucí pohyby. Řídící algoritmus je software, který dodává souboru mechanických součástek, kloubů, ramen, senzorů a kamer možnost pohybu a vykonávání práce, ke které jsou určeny. Protože řízení robotů je komplexní problém, který se neustále rozrůstá o nové technologie, v poslední době jde hlavně o velké využívání umělé inteligence a rozpoznávání prostředí, vznikly systémy, které sdružují všechny potřebné funkce do jednoho celku. Existuje celá řada řídících systémů, které se starají o fungování algoritmů použitých pro řízení robotů. Mezi nimi získal velkou popularitu open source systém ROS, který umožňuje každému upravit jeho funkce podle konkrétních potřeb.

Systém ROS je volně přístupný systém určený primárně pro vývoj, simulování a řízení robotů a rozpoznávání obrazu. O jeho správu se stará společnost Open Robotics, ale na vývoji systému se může podílet kdokoliv. Obsahuje řadu knihoven určených speciálně pro práci s roboty a umožňuje uživatelům vytvářet knihovny vlastní. Teto systém se v současné době využívá v mnoha firmách jako nástroj pro vývoj a řízení robotů. Pracoval jsem s verzí ROS Noetic Ninjemys, která je v současné době nejnovější verzí a má zaručenou podporu nejméně do roku 2025, avšak většina popsaných programů a postupů funguje i na starších verzích systému.

Hlavním úkolem pro tuto bakalářskou práci bylo zjistit možnosti plánování pohybových trajektorií v systému ROS a jejich následné přenesení do externího řídícího systému, kde by je bylo možné využít při řízení manipulátoru. Tímto způsobem pak umožnit řízení sedmiosého manipulátoru přes řídící systém Rexygen.

2 Základní principy systému ROS

Tato část práce se soustředí na vlastnosti, možnosti a využití systému ROS. Rozebere základní funkce systému a několik užitečných knihoven a aplikací spojených s jeho využíváním. Hlavním zdrojem pro tuto sekci byla oficiální dokumentace systému ROS [9] a kniha Programming Robots with ROS [6].

2.1 Instalace systému ROS

Postup instalace ROS je jednoduchý a velmi dobře popsaný v oficiální dokumentaci ROS. Provádí se pomocí příkazového řádku daného operačního systému. V současné době podporuje ROS operační systémy Linux Ubuntu a Debian. Dále ROS umožňuje instalaci na systémy MS Windows 10 nebo Windows 10 IoT Enterprise a Arch Linux, ovšem u těchto operačních systémů se jedná o experimentální provoz a není zajištěna plná funkčnost všech knihoven a aplikací, které jsou součástí systému ROS. V mém případě při instalaci na MS Windows 10 nebylo možné spouštět programy na vizualizaci robotů. Plnou podporu u dříve jmenovaných operačních systémů má až nástupce systému ROS, jímž je ROS2.

2.2 Catkin a ROS package

Program Catkin je sestavovací program pro všechny projekty vytvářené v ROS při vývoji na platformě Linux Ubuntu. Je také prvním programem potřebným pro vytvoření projektu. Jelikož pro fungování všech částí vytvářeného projektu je třeba přesně dodržet předepsanou strukturu souborů v daném projektu, byl vytvořen program Catkin, který se o většinu práce postará za uživatele. Nejprve je ovšem potřeba vytvořit dvě složky. Složku s libovolným názvem, která bude reprezentovat takzvaný Catkin workspace neboli pracovní prostor. Druhá složka, kterou je třeba vytvořit uvnitř složky pracovního prostoru, se musí jmenovat src a bude obsahovat všechny projekty, jinak ji Catkin ani ROS nerozpoznají jako zdrojovou složku programů. Pomocí příkazu catkin_create_pkg <název_balíku> se vytvoří balík projektů, neboli ROS package, jméno lze volit téměř libovolně a za jméno je možné dopsat závislosti na externích knihovnách. Balíky v ROS se dají při-

rovnat k balíkům známým z programovacích jazyků, např java, jedná se o soubor projektů, které mohou navzájem komunikovat, volat se nebo se spouštět. Příkaz catkin_create_pkg vytvoří několik nových složek a souborů uvnitř uživatelem dříve vytvořené složky src. Pro dokončení tvorby balíku je třeba zavolat příkaz catkin_make, což je příkaz pro sestavení projektu. Jelikož v této fázi ještě žádný projekt neexistuje Catkin jen zkontroluje strukturu souborů a dokončí sestavení balíku. Pokud Catkin při sestavování nezahlásí chybu, je balík připraven a vývojář může začít vytvářet projekty. Pro lepší práci s balíkem je ještě dobré přidat balík do prostředí ROS pomocí příkazu source devel/setup.bash, což umožní lepší práci se soubory pomocí funkcí ROS a lehčí spouštění programů.

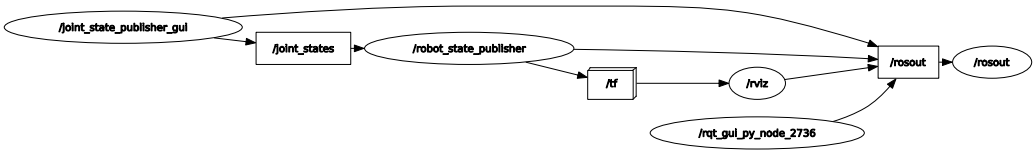
2.3 ROS Node

Pro vytváření jednotlivých souborů je nejprve nutné vytvořit složku projektu, která se bude nacházet v dříve vytvořeném balíku ve složce src. Jedná se ovšem o jinou složku src než ta, která byla popsána v předchozí části, tuto složku vytvořil příkaz catkin_create_pkg uvnitř uživatelem vytvořené složky src při tvoření pracovního prostředí. V tomto bodě je potřeba opět zavolat příkaz catkin_make, který vytvoří další potřebné soubory potřebné pro správný chod projektu.
ROS Node je spustitelný program systému ROS. Jak již bylo psáno v úvodu ROS podporuje v současné době dva programovací jazyky a to C++ a Python. Pro vytvoření nové Node je opět třeba dodržet předepsanou strukturu souborů, vytváříme ji jako soubor s příponou .cpp nebo .py ve složce src nacházející se uvnitř našeho projektu vytvořeného dříve v tomto odstavci, celá cesta k novému souboru by měla vypadat /(název pracovního prostředí)/src/(název balíku)/src/(název programu). Jelikož je ROS systém určený pro běh v reálném čase, běží programy Node většinou v nekonečné smyčce. Délka jednoho průchodu smyčkou neboli tick se nastavuje u každého programu individuálně.

2.4 Ros Topic

Ros Topic jsou prostředí pro sdílení informací mezi jednotlivými programy v projektu. Řídí se danými pravidly, každý Topic má jednu publisher Node neboli program, který zveřejňuje informace nebo zprávy do prostředí a neomezené množství subscriber Node, což jsou programy, které čtou informace z prostředí. Prostředí Topic jsou anonymní, jednotlivé programy neví, kdo

informace čte nebo zapisuje. Pro zápis se používá třída Publisher, která zajišťuje jednak vytvoření Topic, do kterého bude zapisovat, dále pak buffer o velikosti zadané programátorem. Buffer uchovává daný počet zpráv pro pozdější zapsání v případě, že zapisování není dostatečně rychlé nebo je Topic nedostupný a předchází tak ztrátě zpráv. Obdobně třída Subscriber zajišťující čtení informací ze zadaného prostředí obsahuje buffer pro případ, že by přečtené zprávy nebylo možné zpracovat v cyklu, ve kterém byly načteny a uchová je pro pozdější využití. Obecně platí, že jeden program může jak posílat, tak číst zprávy z různých Topic a je tudíž možné vytvářet komplexní komunikační sítě mezi velkým počtem programů.
ROS Topic používá pro komunikaci protokoly TCP/IP a UDP. Ve většině případů se využívá upravená verze TCP/IP nazvaná TCPROS určená výhradně ke sdílení ROS message a service, což jsou formáty zpráv a požadavků, které jsou posílány na prostředí Topic.



Obrázek 2.1: Graf sdílení informací přes různá prostředí Topic

Obrázek ukazuje sdílení zpráv informací mezi programy přes prostředí Topic. Běžící programy jsou ohraničeny elipsami a prostředí Topic jsou ohraničena obdélníky. Na vytvoření tohoto grafu jsem využil program rqt_graph ze stejnojmenného balíku, který se automaticky o vykreslení postará a představuje tedy jednoduchý nástroj pro kontrolu toku informací mezi programy.

2.5 Formát sdílení informací Message

Informace posílané nebo získané z Topic mají formát ROS Message. Každý Topic má svůj datový typ určený pomocí formátu zpráv, se kterými pracuje. Základním tvarem Message jsou základní datové typy, jako jsou například celá čísla, čísla s pohyblivou desetinnou čárkou nebo řetězce znaků. V případě, že má zpráva obsahovat více než jednu hodnotu informace, je možné vytvořit soubor s příponou msg v adresáři balíku, ve kterém se bude využívat. Soubor msg je textový soubor, který obsahuje seznam všech informací, které budou jedním posláním či přečtením zprávy získány. V souboru msg se nemusí vyskytovat jen základní datové typy, ale může obsahovat i seznamy neurčené velikosti, pole nebo jiné dříve zavedené zprávy, lze tak vytvářet

komplexní sady informací pro sdílení například transformačních matic, plánování trajektorií, reprezentaci obrazu v počítačovém vidění atd. V souboru msg je také možné vytvářet konstanty a vlastní datové typy.

2.6 roscore

Jádro systému ROS, neboli roscore, je soubor programů nutných pro spuštění a běh všech programů systému ROS. Jádro se stará o komunikaci mezi jednotlivými programy a obstarává běh prostředí Topic. Zároveň spouští program MASTER, který se stará o správné zapisování a čtení informací z Topic a umožňuje programům se navzájem lokalizovat a komunikovat pomocí peer-to-peer spojení. Další důležitou součástí jádra je server parametrů. Tento server je prostor pro ukládání a sdílení různých proměnných, většinou ve formě parametrů, které programy při svém spuštění potřebují. K tomuto serveru mají přístup všechny programy pomocí protokolu REST API. V případě, že má být parametr dostupný pouze jednomu programu, je třeba při jeho založení před jeho jméno napsat znak $\sim$. Současně se spustí jedno prostředí Topic se jménem rosout, které slouží jako mechanismus hlášení stavu systému na konzoli. Zprávy do něj přidává speciální program rosout, který neustále kontroluje stav programů i samotného systému.

2.7 Spouštění programů

Po vytvoření programu je nutné před spuštěním provést sestavení. Každý balík obsahuje soubor CMake, který obsahuje informace o názvech spustitelných programů, které se mají sestavit. Dále se v něm specifikují zprávy, se kterými by programy měli pracovat a knihovny, které budou programy využívat. Poté se použije sestavovací program Catkin a již dříve zmiňovaný příkaz catkin_make, který sestaví spustitelné programy.
Aby bylo možné spustit programy systému ROS, musí se nejprve spustit jádro systému roscore. Následně je možné spustit program pomocí příkazu rosrun. Jako první dva argumenty se uvádí jméno balíku a jméno programu, dalšími parametry mohou být závislosti na jiných programech nebo parametry pro spuštění. Při spouštění pomocí příkazu rosrun je nutné každý z programů spouštět v samostatném okně příkazového řádku, jelikož po spuštění se příkazový řádek, ve kterém program běží používá jako konzole pro vstupy a výstupy. Pro ukončení běhu programu se používá klávesová zkratka Ctrl+c.

Pokud je třeba spustit více programů najednou, což je častější případ při používání systému ROS, využívají se spouštěcí soubory s příponou launch. Jedná se o soubory ve formátu XML, které obsahují seznam spouštěných programů s názvy balíků, programů a jejich parametrů. Dále mohou obsahovat parametry, které se při spuštění zapíší do serveru parametrů, zmiňovaného v minulé části. Uvnitř spouštěcího souboru je i možné používat logické operátory if a unless. Operátor if má stejnou funkcionalitu jako ve většině programovacích jazyků, obsah jeho těla se použije jen pokud je jeho podmínka pravdivá. Operátor unless by se dal přeložit jako jestliže neplatí a jeho vyhodnocení podmínky tedy pracuje přesně opačně než u operátoru if. Při použití spouštěcího souboru je třeba počítat s tím, že všechny spuštěné programy mají jen jeden společný konzolový vstup a výstup. Pro ukončení všech spuštěných programů přes spouštěcí soubor se opět používá klávesová zkratka Ctrl+c použitá v konzoli, kde byly programy spuštěny, jako u spouštění přes příkaz rosrun.

Ukázka kódu 2.1: Spouštěcí soubor Launch.

```
<launch>
        <param name="robot_description" command=
    "$(find xacro)/xacro --inorder '$(find
        beginner_tutorials)/urdf/robot1c.xacro'"/>
            <node name="robot_state_publisher"
            pkg="robot_state_publisher"
            type="robot_state_publisher"/>

        <node name="rviz" pkg="rviz" type="rviz"
        args="-d $(find beginner_tutorials)
            /launch/config.rviz"/>

        <node name="joint_state_publisher_gui"
            pkg="joint_state_publisher_gui"
            type="joint_state_publisher_gui"/>
</launch>
```

V ukázce je spouštěcí soubor, který nahraje jeden parametr na server parametrů a následně spustí tři programy Node.

2.8 Spolupráce programů běžících na různých zařízeních

Jak již bylo naznačeno v kapitole o prostředí Topic, systém ROS používá pro komunikaci protokoly TPCROS. Nejen že je možné dostávat tímto způsobem zprávy z jiných zařízeních, je také možné spouštět programy na různých zařízeních jako by běžely na jednom stroji. V tomto případě se na jednom zařízení spustí jádro roscore, které bude pro všechny zařízení společné. V ostatních zařízeních je pak jen třeba specifikovat parametr ROS_MASTER_URI, který udává adresu, kde je jádro systému dostupné. Tato funkce najde uplatnění například při potřebě synchonizovaně řídit několik robotů najednou, kde na každém robotu běží řídící program a na nějakém dalším zařízení běží jádro a program vytvářející trajektorie.

3 Možnosti modelování robotů a plánování jejich pohybu v systému ROS

V této části se práce bude soustředit na knihovny a programy používané k práci s roboty, jejich modelování, zobrazování a simulace. Také se bude zabývat problematikou plánování pohybu robotů.

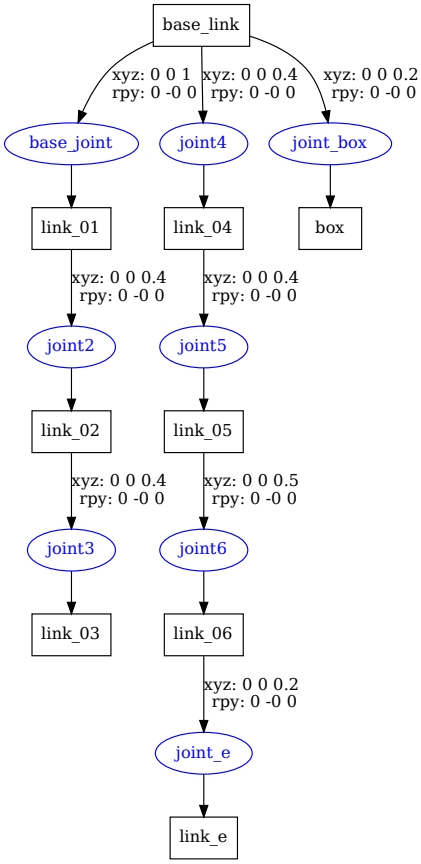
3.1 Knihovna tf

Knihovna tf je jedna z nejdůležitějších knihoven systému ROS pro práci s roboty. Hlavními funkcemi této knihovny je zavádění souřadných systémů a přechodů mezi nimi v trojrozměrném prostoru a stará se o prostředí Topic tf, které slouží pro sdílení informací o poloze mezi programy. Zavádí formu definování souřadných systémů ve stromové struktuře. Vždy existuje jeden souřadný systém světa, který je většinou pevný a neměnný. Od něho se zavádějí další systémy, jako jsou například ramena manipulátoru, poloha dronu nebo obecně umístění předmětů v prostoru. Každý souřadný systém je zadán vzhledem ke svému předchůdci posuvem ve třech osách a Eulerovými úhly nebo kvaternionem. Při transformacích mezi souřadnými systémy se nejprve provedou posuny ve všech třech osách a následně rotace podle fixovaných os. Při rotaci podle Eulerových úhlů se rotuje o zadané úhly podle původních os v pořadí X, Y a nakonec Z. Kvaterniony nejsou zadány jednoznačně, každou rotaci je možné reprezentovat dvěma kvaterniony, které jsou navzájem inverzní. Všechny posuny se uvádějí v metrech a rotace v radiánech. Knihovna tf obsahuje také funkce, které umožňují převod mezi Eulerovými úhly a kvaterniony.

3.2 Tvorba modelu robotu

Pokud chceme využít systém ROS pro řízení robotů, nebo jako v případě této práce, pro plánování trajektorií, je nutné nejprve vytvořit model robotu, se kterým budeme pracovat. ROS používá pro popis robotů soubory URDF, což je Universal Robotic Description Format, neboli univerzální formát popisu robotů. Soubor má formu XML a popisuje trojrozměrný mo-

del daného robotu. U každého robotu je třeba specifikovat jeho název, aby ostatní programy věděli, se kterým modelem zrovna pracují. Jako většina popisů i URDF popisuje roboty pomocí ramen (Link) a kloubů (Joint), ale může obsahovat i popis materiálů, ze kterých je robot vyroben. Popis má stromovou strukturu, kde ramena jsou vrcholy a klouby jsou hrany.



Obrázek 3.1: Graf ukazující stromovou strukturu modelu URDF

3.2.1 Ramena v URDF

Ramena jsou viditelné části robotu. Každé rameno musí mít jedinečný název v rámci jednoho modelu, aby je bylo možné rozeznat. Ramena mají tři hlavní tagy a to jsou visual, neboli viditelná část, collision, neboli definice tvaru ramene, ve kterém může dojít ke kolizi s jiným ramenem a innertia, což specifikuje mechanické vlastnosti ramene. Popis ramene nemusí obsahovat všechny tři tagy, ale při simulaci robotu je pak výsledná simulace méně přesná. Při jednoduchém modelování ve formě ručního psaní URDF souboru

se pro vytváření grafické a kolizní části využívá tag geometry, který obsahuje popis základních geometrických těles, jako jsou válec, kvádr nebo koule. Dále je možné u obou částí uvést polohu v prostoru vzhledem k souřadnému systému světa nebo v častějším případě vzhledem k předchozímu kloubu. Poloha se uvádí v již známém formátu z knihovny tf, tedy posun ve třech osách a Eulerovy úhly nebo kvaternion. Je tedy možné, že grafická část se nachází jinde než kolizní nebo má jiné rozměry. Další vlastností ramen může být materiál a jiné mechanické vlastnosti jako tření ramen v případě kolize, koeficient tuhosti a tlumení kmitů. Tyto vlastnosti se využijí při simulaci v programu Gazebo [2], který na základě těchto informací dokáže vytvořit simulaci kolizí ramen a možné kmitání ramen při pohybu robotů. Této funkcionality je vhodné využít v případě, že se robot pohybuje vysokou rychlostí nebo se během pohybu jeho ramena k sobě přibližují a může dojít k jejich dotyku, abychom ověřili chování robotu v těchto situacích.

Ukázka kódu 3.1: Rameno robotu vytvořené v modelu URDF.

```
<link name="link_01">
        <visual>
            <material name="orange"/>
            <origin xyz="0 0 0.2" rpy="0 0 0"/>
            <geometry>
                <cylinder length="0.4" radius="0.4"/>
            </geometry>

        </visual>
        <collision>
                <origin xyz="0 0 0.2" rpy="0 0 0"/>
            <geometry>

                <cylinder length="0.35" radius="0.4"/>
            </geometry>
        </collision>

</link>
```

Na ukázce je vidět tvorba ramene robotu s vizuální a kolizní částí ve tvaru válců o dané výšce a poloměru podstavy.

3.2.2 Klouby v URDF

Klouby fungují jako spojnice mezi rameny. Nemají grafickou podobu a nejsou tedy ve vizualizacích viditelné. Každý kloub musí mít, stejně jako rameno, své jedinečné jméno. Jeho další atribut je typ kloubu. URDF zná čtyři základní typy kloubů. Nejjednodušší je fixovaný kloub, který představuje pevné spojení dvou ramen bez možnosti pohybu. Další tři klouby jsou pohyblivé. Posuvný (prismatic) kloub umožňuje pohyb v jedné ose, planární (revolute) kloub se pohybuje ve dvou osách a kloub sférický (floating), který se může volně rotovat kolem všech třech os. Dva nejdůležitější tagy kloubů jsou parent a child, které odkazují na dvě ramena, která jsou tímto kloubem spojena. U kloubů se opět specifikuje jejich poloha, ale určuje se vzhledem k souřadnému systému prvního z ramen připojeného ke kloubu. Pomocí tagu axis se u posuvných a planárních kloubů určí, kolem které osy se bude kloub pohybovat, v případě že tato informace není pro nějaký kloub specifikována, bude se otáčet kolem osy x. U pohyblivých kloubů je také nutné nastavit limity přes tag limit. Jeho atributy jsou maximální moment síly (effort), maximální možnou rychlost (velocity) a maximální a minimální limity polohy, do kterých se kloub může dostat. Dalšími možnými atributy kloubů jsou tření nebo tlumení, pokud tyto atributy nejsou zadány, je jejich hodnota nastavena na 0.

Ukázka kódu 3.2: Planární kloub vytvořený v modelu URDF.

```
<joint name="base_joint" type="revolute">
        <axis xyz="0 0 1"/>
        <limit effort="1000" lower="-3.14"
            upper="3.14" velocity="0.5"/>
        <origin xyz="0 0 1" rpy="0 0 0"/>
        <parent link="base_link"/>
        <child link="link_01"/>
</joint>
```

V ukázce je naznačena tvorba planárního kloubu, který rotuje kolem osy z. Má specifikovaná pohybová omezení a dvě ramena, která spojuje.

3.2.3 Jazyk xacro

Pro lepší práci s modelem a omezení opakování kódu byl pro URDF vyvinut macro jazyk s názvem xacro. Tento jazyk umožňuje vytvářet macra, definovat konstanty nebo používat matematické výpočty, které samotné URDF nepodporuje. Pro správné fungování jazyka xacro je třeba místo přípony

souboru urdf použít příponu xacro a k tagu robot, který označuje začátek popisu robotu, přidat atribut
$xmlns : xacro = "http : //ros.org/wiki/xacro"$. Hlavním účelem jazyka je tedy zjednodušení popisu a vyvarování se zbytečného opakování kódu. Velmi užitečnou součástí jsou také matematické konstanty a to hlavně π, které je užitečné při zadávání úhlů, které jsou vždy v radiánech. Xacro se dá taky využít pro vytváření vlastních geometrických obrazců, ať už úpravou existujících obrazců nebo jejich spojováním do složitějších objektů.

Ukázka kódu 3.3: Deklarace konstanty a macra na tvorbu válce pomocí xacro a jeho použití.

```
<xacro:property name="my_radius" value="2.1" />

<xacro:macro name="easy_cylinder"
        params="name␣radius␣length␣*origin␣*material">
<link name="${name}">
        <visual>
                <xacro:insert_block name="origin" />
                <geometry>
                        <cylinder radius="${radius}"
                            length="${length}" />
                </geometry>

                <xacro:insert_block name="material" />
        </visual>

        <collision>
                <xacro:insert_block name="origin" />
                <geometry>
                        <cylinder radius="${radius}"
                            length="${length}" />
                </geometry>
        </collision>
</link>
</xacro:macro>

<xacro:easy_cylinder name="base_link"
        radius="0.10" length=".30">
        <origin xyz="0␣0␣.15"/>
        <material name="black">
                <color rgba="0␣0␣0␣1"/>
        </material>
</xacro:easy_cylinder>
```

Na ukázce je vidět deklarace konstanty pomocí jazyka xacro. Následně je vytvořeno makro pro jednoduší vytváření ramene ve tvaru válce a ukázáno použití tohoto makra.

3.2.4 Získání URDF z modelovacích programů

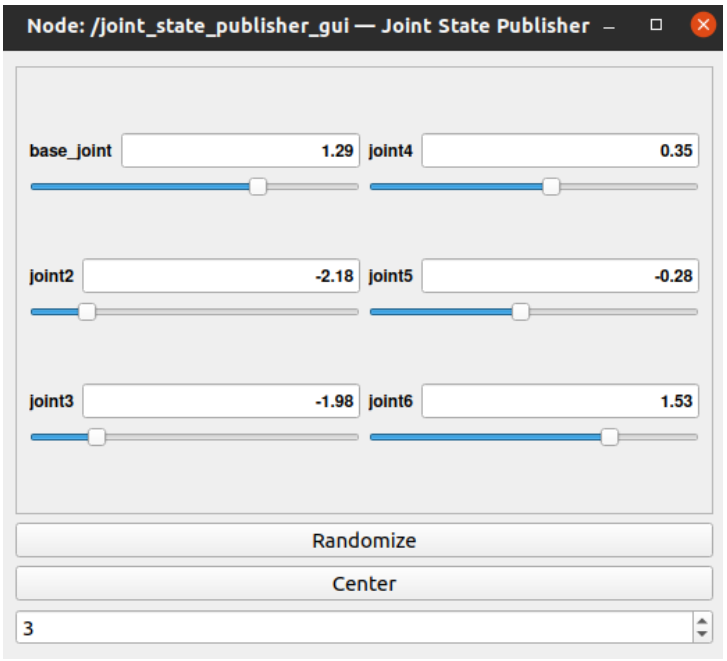
Jelikož většina standardních robotů není složena pouze ze základních trojrozměrných objektů, není vhodné takto vytvářet jejich modely. Jako alternativa se nabízí využití modelů z jiných systémů či programů, které jsou určeny na tvorbu 3D mechatronických modelů. Jako příklad mohu uvést programy SolidWorks a Creo, do kterých je možné doinstalovat rozšíření, pomocí kterých lze model exportovat ve formátu URDF. Další možností, jak získat přesný model robotu, je stáhnutí popisu z dostupných zdrojů na internetu.
Stejně tak jako se dá model URDF získat z jiných programů, dá se do různých programů importovat, příkladem je Matlab a jeho simulační část Simscape Multibody.

3.3 Knihovna robot_state_publisher

Jelikož soubor URDF je za běhu programu neměnný a není možné ani vhodné do něho zapisovat aktuální polohu kloubů a ramen, využívá se spojení knihoven tf a robot_state_publisher. Hlavní částí knihovny robot_state _publisher je stejnojmenný program, který je schopen načíst model robotu URDF ze serveru parametrů a na jeho základě vypočítávat aktuální polohy kloubů a ramen, které poskytuje programům knihovny tf. Aktuální polohu částí robotu nepočítá pouze z modelu, ale může přijímat i vstupní data o poloze kloubů jako zprávy z knihovny tf, kam tyto informace nahrávají jiné programy. Je nutné tento program spustit při jakékoliv práci s modelováním a plánováním pohybu robotů.

3.4 Program joint_state_publisher

Joint_state_publisher je jeden z programů, které mohou nahrávat zprávy o poloze kloubů přes knihovnu tf do programu robot_state_publisher. Tento program najde využití hlavně při vizualizaci naplánovaných trajektorií, jelikož při simulacích nahrává tyto zprávy simulační program a při reálném řízení ho zastoupí senzory. Jeho variace s grafickým rozhraním Joint_state _publisher_gui se dá využít pro ruční nastavování kloubových souřadnic.



Obrázek 3.2: Program joint_state_publisher_gui.

Na obrázku je podoba programu joint_state_publisher_gui. Je vidět, že umožňuje nastavování kloubových souřadnic jednotlivých kloubů, vytvoření náhodné polohy a vycentrování všech kloubů do středu jejich rozsahu možné polohy. Tento program nijak nekontroluje možné kolize a proto při použití náhodné polohy často generuje nedosažitelné stavy robotu.

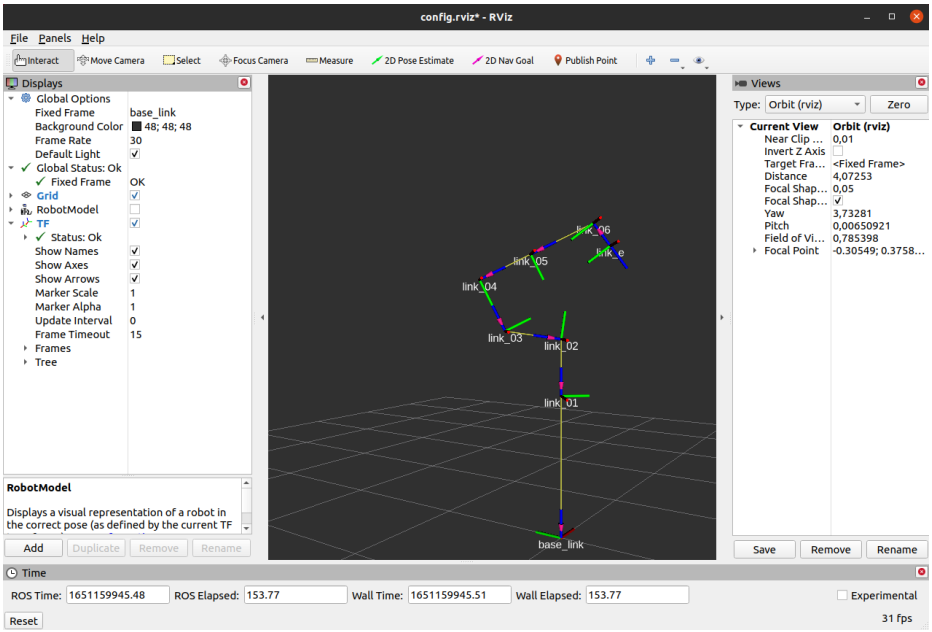
3.5 Vizualizační program rviz

Při plné instalaci systému ROS se automaticky nainstaluje i program rviz, který slouží k vizualizaci trojrozměrných modelů a objektů. Jedná se o univerzální nástroj, který umí spolupracovat s programy systému ROS, zobrazovat je a interagovat s nimi. Vykreslování se zajišťuje pomocí displejů, které se vybírají z nabídky displejů. Mezi základní displeje jsou Grid, neboli mřížka, ta může být vykreslena buď na 2D plochu nebo do celého 3D prostoru. Dalším základním displejem je Axis, který vykreslí osy světa podle souřadného systému světa, který je zvolen uživatelem. Pomocí displejů Point a Polygon je možné přidávat do výkresu vlastní objekty, kde Point vykreslí bod v podobě malé koule a Polygon vykreslí obecný trojrozměrný mnohoúhelník zadaný pomocí jednotlivých hran. Tyto objekty jsou vykreslovány podle dat, která rviz získá ze zpráv z prostředí geometry_msgs. Jelikož při každém spuštění programu se program pustí v základním nastavení, je vhodné

si po nastavení potřebných displejů toto nastavení uložit do konfiguračního souboru. Do konfiguračního souboru se také uloží nastavení použitých nástrojů, kamer s jejich počátečním pohledem a senzorů. Při příštím spuštění je možné přidat ke spouštěcímu příkazu jako parametr právě konfigurační soubor, nebo po otevření programu rviz v levém horním rohu v nabídce File ručně konfigurační soubor přidat.

3.5.1 Zobrazování robotů

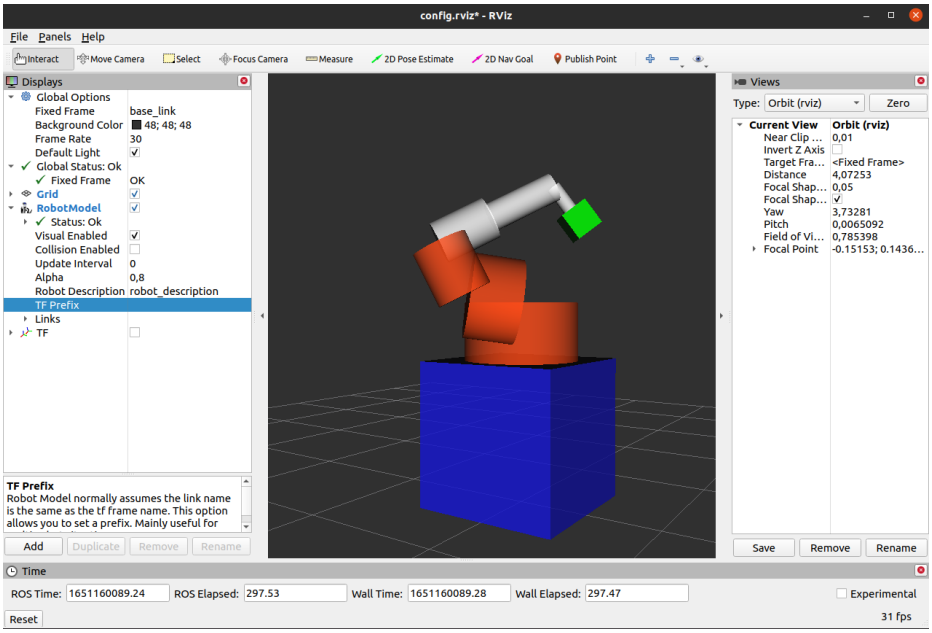
Pro zobrazování robotů je vhodné použít displeje TF a RobotModel. Displej TF čte zprávy od programů knihovny tf, které popisují jednotlivé souřadné systémy ve stromové struktuře a vykreslí je v podobě jejich os do vykreslovaného prostoru. Umožňuje pomocí nastavitelných parametrů zobrazit nejen osy souřadného systému, ale také jeho název nebo šipky spojující dva po sobě jdoucí souřadné systémy ve stromové struktuře. Umožňuje také individuální nastavení každého systému, což obsahuje jeho viditelnost a zobrazení jmen otcovského a dceřiného souřadného systému.



Obrázek 3.3: Použití displeje TF v programu rviz.

Na obrázku je ukázka použití displeje TF, který vykreslil všechny souřadné systémy, které v dané době registruje knihovna tf, včetně jejich jmen a spojení mezi nimi.

Displej RobotModel je určen k vykreslování grafické podoby robotu. Potřebuje tedy model robotu v souboru URDF, který najde na serveru parametrů a zprávy z knihovny tf, podle kterých vykreslí aktuální polohy jednotlivých kloubů a ramen. RobotModel umí vykreslovat ze souboru URDF jak objekty s tagem visual, tak i objekty s tagem collision, je tedy možné pozorovat jak visuální podobu robotu, tak i zkoumat možné kolize. Umožňuje stejně jako displej TF zobrazovat osy souřadných systémů. Dalšími nastavitelnými parametry jsou nastavení průhlednosti ramen a obnovovací frekvence, která udává, jak často se bude model aktualizovat.



Obrázek 3.4: Použití displeje RobotModel v programu rviz na manipulátor s šesti stupni volnosti.

Na obrázku je ukázáno využití displeje RobotModel pro vykreslení modelu šestiosého manipulátoru.

Pokud máme nahraný model robotu na serveru parametrů a vykreslenou jeho podobu pomocí displeje RobotModel, můžeme začít s robotem pohybovat. Aby bylo možné měnit polohy souřadných systémů, musíme nejprve spustit program robot_state_publisher. Následně můžeme využít buď program joint_state_publisher_gui pro jednoduché upravování kloubových souřadnic, nebo vytvořit vlastní program, který bude posílat námi zvolené kloubové souřadnice pro robot_state_publisher.

3.6 Simulační program Gazebo

Program Gazebo [2] je určen především k simulaci reálného světa. Je automaticky nainstalován při plné instalaci systému ROS, ale je jej možné stáhnout, nainstalovat a používat i samostatně. Gazebo je vytvořen tak, aby mohl jak přijímat tak vysílat zprávy ROS Message, které si vnitřně přetypuje na své formáty. Gazebo tedy dokáže simulovat jak vstupy, tak výstupy reálného zařízení.
Stejně jako program rviz, také Gazebo využívá pro vykreslení vizuálních částí robotu soubory URDF, ze kterých si ale vytvoří vlastní soubor typu SDF. Při spuštění simulace se stejným souborem, který byl použit při vizualizaci, ale zjistíme, že model pod působením gravitace v simulovaném světě spadne na zem a nebude možné s ním pohybovat. V tuto chvíli dokonce ani Gazebo nekomunikuje s programy systému ROS, pouze simuluje obsah souboru URDF. Aby bylo možné používat Gazebo společně se systémem ROS je potřeba spustit simulační program přes příkaz *rosrungazebo_rosgazebo*, který umožní systému ROS komunikovat se simulovaným světem. Následně je ještě potřeba speciálně upravit soubor URDF, aby bylo možné simulovaný svět ovládat. Nejprve se musí doplnit URDF o modul *gazebo_ros_control* z knihovny *libgazebo_ros_control.so*, který umožňuje vzájemnou komunikaci programů. Následně je potřeba specifikovat kontrolery. Kontrolery jsou speciální parametry simulace ve formátu *yaml*. Mohou slučovat řízení několika kloubů, např. při řízení robotu na kolech nebo maximální rychlosti a zrychlení jednotlivých kloubů. Poslední věcí, která chybí v souboru URDF, jsou Transmission neboli převody, které specifikují chování kloubu pro simulaci. Určují typ aktuátoru, typ vstupu a odkazují i na kontrolery.

Ukázka kódu 3.4: Jeden z převodů vytvořený pro model šestiosého manipulátoru.

```
<transmission name="trans_base_joint">
        <type>transmission_interface/SimpleTransmission
        </type>
        <joint name="base_joint">
            <hardwareInterface>hardware_interface
            /EffortJointInterface
            </hardwareInterface>
        </joint>
        <actuator name="base_joint_motor">
           <hardwareInterface>hardware_interface
            /EffortJointInterface</hardwareInterface>
```

```
            <mechanicalReduction>1</mechanicalReduction>
        </actuator>
</transmission>
```

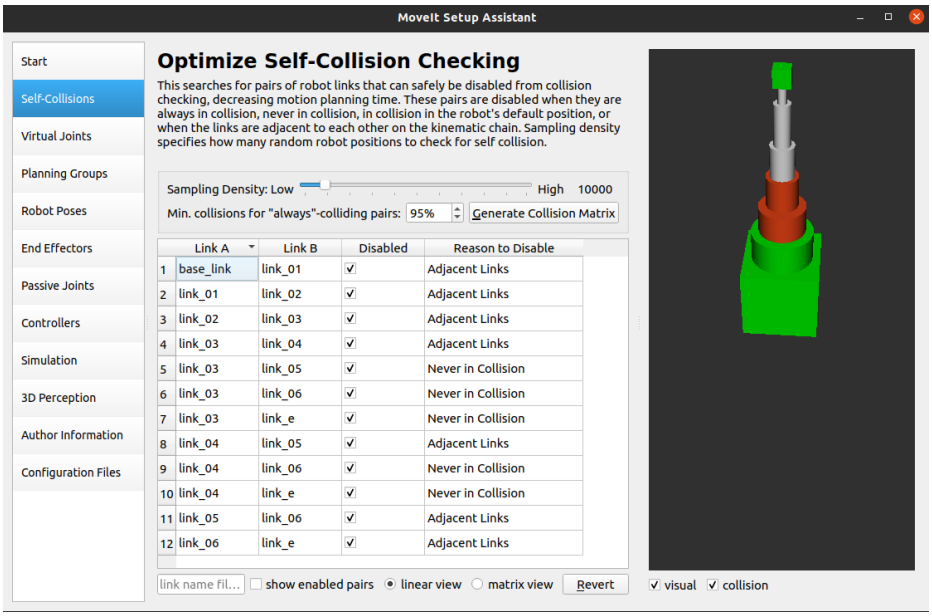
Na ukázce je jeden z převodů Transmission vytvořený pro jeden z kloubů šestiosého manipulátoru.

3.7 Program pro plánování trajektorií MoveIt!

Program MoveIt! [5] najde hlavní využití při plánování trajektorií pro pohyb robotů řízených pomocí systému ROS. MoveIt! se na rozdíl od předchozích programů neinstaluje jako součást systému ROS, ale je jej třeba doinstalovat. Program je volně přístupný a spravovaný společností PickNik Robotics. MoveIt! obsahuje tři možnosti rozhraní, prvním je balík move_group_interface určený pro programy psané v jazyce C++, druhým je balík MoveIt!_commander užívaný v programech vytvořených v jazyce Python a třetím je grafické rozhraní Motion Planning plugin, které využívá program Rviz a umožňuje grafické plánování pohybů.

3.7.1 Konfigurace modelu pro program MoveIt!

Aby bylo možné používat plánování trajektorií pomocí programu MoveIt!, je nutné nejprve vytvořit model robotu ve známém formátu URDF. Následně je třeba provést konfiguraci modelu, aby MoveIt! dostal všechny potřebné informace. Nejjednodušším způsobem je použít program Setup Assistent, který pomocí grafického rozhraní provede uživatele celou konfigurací. Jediné, co program potřebuje, je model robotu ve formátu URDF, popřípadě již dříve vytvořenou konfiguraci. V současné době podporuje i formát XACRO, ale není jisté, že bude s tímto formátem plně fungovat. Po vložení modelu program vykreslí grafickou podobu modelu a je možné vykreslit i grafickou kolizní podobu modelu. Dalším krokem je vytvoření takzvané kolizní matice. Tato matice obsahuje informace o možných kolizích ramen robotu a určuje, která ramena se sebou nemohou nikdy dojít do kolize, čímž zrychluje následné výpočty trajektorií, protože zkrátí kontrolu kolizí pří plánování pohybu. Tato matice se generuje automaticky na základě uživatelem zadaného počtu náhodně vygenerovaných poloh modelu.



Obrázek 3.5: Program Setup Assistant a vzájemné polohy ramen získané z kolizní matice.

Dalším krokem je možnost přidání virtuálních kloubů, tento krok se většinou využívá k definování souřadného systému světa, pokud nebyl definován již v modelu URDF. Následuje vytvoření plánovacích skupin. Tyto skupiny rozdělují model na jednotlivé části, v případě manipulátoru většinou na samotný manipulátor a koncový efektor. U robotů s více rameny se využije rozdělení na skupiny pro definování jednotlivých ramen. Pro každou skupinu je možné vybrat vlastní řešení inverzní kinematické úlohy nebo vybrat z numerických solverů. Po vytvoření skupin je možné přidat známé polohy robotu pomocí kloubových souřadnic. Je tedy možné takto definovat domovskou pozici a pracovní pozice robotu. Následně se definují koncové efektory. Je možné model doplnit o pasivní klouby, které není možné řídit a senzory. Posledním krokem je doplnění kontrolerů a automatické vytvoření URDF souboru pro simulace v programu Gazebo. Před konečným vytvořením konfigurace uživatel zvolí, které konfigurační soubory chce vytvořit. Automaticky jsou vybrány všechny soubory a není doporučené toto nastavení měnit. Následně se vytvoří konfigurace modelu jako samostatný balík programů ROS.

3.7.2 Solvery pro inverzní kinematickou úlohu programu MoveIt!

Program MoveIt! využívá jako základní plánovací algoritmus framework OMPL [11], což je zkratka pro Open Motion Planning Library. Jedná se o soubor algoritmů, které dokáží numericky řešit inverzní kinematickou úlohu a určit tedy trajektorii pohybu robotu mezi počáteční a koncovou polohou. Algoritmy OMPL jsou určeny pouze pro plánování poloh, nepočítají rychlosti ani zrychlení robotů. Tyto hodnoty určuje zpětně po vypočtení polohové trajektorie program MoveIt!, který dopočítá rychlosti a zrychlení takové, aby umožnily průchod trajektorií a splňovaly omezení jejich maximální velikosti. MoveIt! také automaticky volí časové rozmezí mezi jednotlivými body trajektorie tak, aby vytvořené rychlosti a zrychlení odpovídali naplánované polohové trajektorii. Není tedy možné zadat pevné časové rozmezí mezi jednotlivými body v trajektorii. Další algoritmy podporované programem MoveIt! jsou CHOMP, STOMP a TrajOpt plánovače.

4 Plánování standardních a komplexních trajektorií

Pro plánování trajektorií robotu jsem využil program MoveIt!, který lze využít pro plánování, jak standardních, tak i komplexních trajektorií. Mezi standardní trajektorie plánované programem MoveIt! patří pohyb po přímce a pohyb mezi dvěma body po ideální trajektorii, kterou zvolí solvery programu MoveIt!. Pro vytváření komplexní trajektorie jsem použil možnost přidání objektů do prostředí, ve kterém se robot pohybuje a se kterými nesmí dojít do kolize.

4.1 Grafické rozhraní Motion Planning

MoveIt! umožňuje kromě programovacích rozhraní využít i grafické rozhraní v programu Rviz, které vytváří rozšíření Motion Planning. V programu Rviz se zobrazí grafická podoba modelu. Následně uživatel vybere počáteční a koncovou polohu robotu. Tento výběr je možný buď z grafického modelu přetáhnutím koncového efektoru, ručním zadáním kloubových souřadnic nebo výběrem jedné ze zadaných poloh vytvořených při konfiguraci modelu. Poté je možné kliknout na tlačítko Plan a program MoveIt! se pokusí vytvořit vhodnou trajektorii. Tuto trajektorii jednak zobrazí v grafické části programu Rviz a také ji odešle z pravidla do prostředí /move_group/display_planned_path, pokud nebylo uživatelem zvoleno jiné prostředí. Z tohoto prostředí je možné naplánovanou trajektorii číst jiným programem, nebo ji vypsat příkazem rostopic echo /move_group/display_planned_path, který vypíše všechny údaje naplánované trajektorie do konzole.

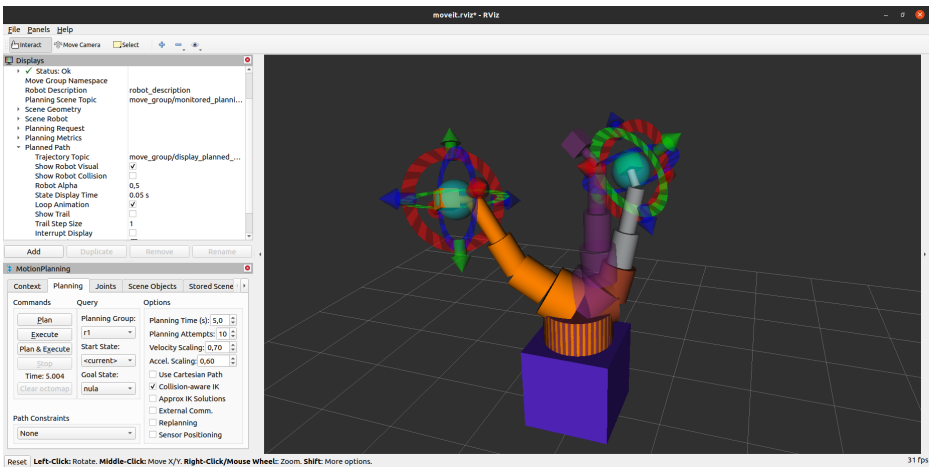
Ukázka kódu 4.1: Ukázka jednoho z bodů naplánované trajektorie.

```
positions: [0.7242309884419814, -0.29985170863695587,
0.553178518469, -0.2917391699669, -0.7592752168036,
0.0709591693063]

velocities: [0.0012528229624786, 0.300088835924, -0.5,
0.266669239546, 0.06562397702, -0.127755180947]

accelerations: [-0.0004943547763, -0.076335598477, 0.0,
0.025254479497667, -0.07346180608763, 0.11487851436177]
```

```
effort: []
time_from_start:
secs: 3
nsecs: 249975835
```

Jak vidíme v ukázce, MoveIt! plánuje trajektorii včetně rychlostí a zrychlení jednotlivých kloubů, ale je nutné brát v potaz časové rozmezí mezi dvěma body trajektorie, které volí program MoveIt! a z mých zkušeností vím, že tyto časová rozmezí bývají i větší než jedna vteřina. Poslední dva řádky ukázky obsahují informaci o čase od začátku naplánované trajektorie v celých sekundách a nanosekundách.

Dalšími nastavitelnými parametry v grafickém rozhraní je, mimo jiné, možnost plánování trajektorie po přímce, či striktní vyhýbání kolizím. V případě, že se programu nepodaří nalézt proveditelnou přímkovou trajektorii, zobrazí jen část trasy do bodu, do kterého je přímkový pohyb možné uskutečnit. Je možné také upravit maximální povolené kloubové rychlosti a zrychlení, jako násobek těchto limitů zadaných v souboru URDF v rozmezí 0 až 1. V případě, že trajektorie vygenerovaná solverem vybraným při konfiguraci modelu není vhodná, lze v nastavení plánující solver dočasně změnit. Obecně však grafické rozhraní obsahuje méně možností pro plánování trajektorie než rozhraní programová.



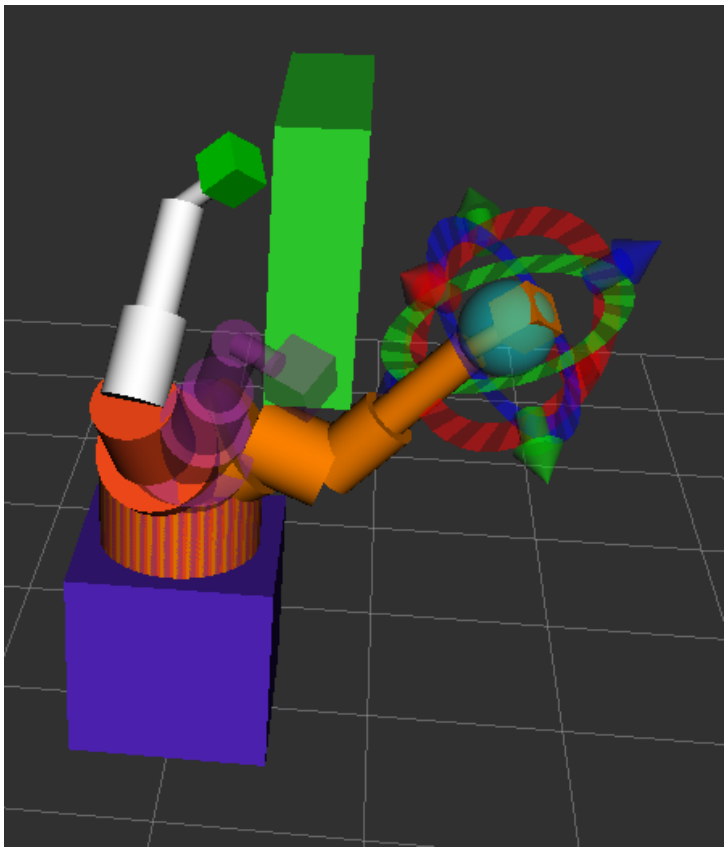
Obrázek 4.1: Využití grafického rozhraní programu MoveIt! v programu Rviz.

Na obrázku vidíme v levém dolním rohu grafické rozhraní pluginu Motion Planning, které obsahuje veškeré údaje o plánované trajektorii a provádí i samotné plánování. V grafické části vidíme tři vizualizace modelu. Různobarevný model zobrazuje počáteční polohu manipulátoru, fialový model

ukazuje vzorový průchod trajektorií a oranžový model je koncová poloha robotu. Počáteční i koncová poloha se vykresluje včetně ovládacích prvků, které se používají pro grafické nastavení těchto poloh.

4.1.1 Plánování komplexní trajektorie v grafickém rozhraní

Pro plánování komplexnějších trajektorií v grafickém rozhraní MoveIt! se využívá především možnost přidání kolizních objektů do okolí modelu. Jelikož jsou tyto objekty jak vizuální, tak mohou způsobit i kolize s modelem při procházení plánované trajektorie, pokusí se MoveIt! najít trasu, která by se měla těmto objektům vyhnout. Nejjednodušším způsobem, jak přidat nějaký objekt do modelované scény, je využít možnost vytvoření jednoduchých těles přímo v grafickém rozhraní pod záložkou Planning scene. Nejprve je nutné zvolit druh objektu, na výběr je kvádr, koule, válec a kužel. Následně se zadají rozměry objektu a je možné jej vložit do plánovací scény. Další možností je importovat model objektu ve formátu URDF. Pro nastavení polohy a orientace objektu je možné použít ovládací prvky přímo na vykresleném objektu nebo nastavit číselné hodnoty v záložce, kde byl vytvořen/importován. Popřípadě lze takto přidaný objekt připevnit ke koncovému efektoru robotu a plánovat tak například trajektorii manipulátoru, jehož úkolem je přeprava nějakého produktu.
Po přidání všech kolizních objektů se opět jen nastaví počáteční a koncová poloha robotu a stiskne se tlačítko Plan a dojde k vytvoření trajektorie pomocí solveru OMPL [11] a jednoho z jeho algoritmů. Já jsem používal algoritmus RRT*[3]. V případě, že se vytvořená trajektorie příliš přiblíží koliznímu objektu, nebo je jiným způsobem nevyhovující, může uživatel zaškrtnout kolonku Collision-aware IK v rozhraní Motion Planning. Po zaškrtnutí se při příštím plánování trajektorie MoveIt! pokusí vytvořit lepší trasu z hlediska rizika kolize na úkor délky trajektorie. Při plánování těchto trajektorií se většinou nutné zvýšit povolený maximální čas výpočtu trasy v rozhraní plánovače, jelikož se provádějí složitější výpočty a za přednastavený čas nemusí program dojít k řešení.



Obrázek 4.2: Plánování trajektorie kolem kolizního objektu.

Na obrázku je ukázka plánování pohybové trajektorie, na které je mezi počáteční a koncovou polohu manipulátoru přidán kolizní objekt ve formě zeleného kvádru.

4.1.2 MoveIt Commander

Program MoveIt Commander přestavuje kompromis mezi grafickým a programovým rozhraním plánovače. Po spuštění grafického rozhraní a programu rviz může uživatel spustit také MoveIt Commander. Tento program představuje možnost ovládání grafického rozhraní pomocí příkazů psaných do konzole a do jisté míry doplňuje chybějící funkce, které jsou zavedeny v rozhraních programových, jako je například nastavení tolerance dosažení cílové polohy nebo zadání polohy a natočení koncového efektoru jako cíl plánované trajektorie.

4.2 MoveIt! a jeho C++ a Python rozhraní

Základní způsob plánování trajektorií pomocí programu MoveIt! jsou rozhraní spojující ROS a MoveIt! vyvinutá pro jazyky C++ a Python pomocí třídy MoveGroup. Rozhraní se využívá v programech ROS Node. Oproti grafickému rozhraní umožňují komplexnější nastavení plánované trajektorie. Umožňují dva způsoby zadání požadované koncové polohy robotu. Prvním je zadání kloubových souřadnic všech kloubů a druhým a užitečnějším je poloha a orientace koncového efektoru vzhledem k základnímu souřadnému systému. Roboty s více stupni volnosti mohou mít více možných konfigurací při dosažení požadované polohy koncového efektoru. Jelikož MoveIt! používá numerické solvery a algoritmy, vybere zpravidla to řešení inverzní kinematické úlohy, které nalezne jako první. Naplánovaná trajektorie se stejně jako u grafického rozhraní nahraje do prostředí Topic /move_group/display_planned_path. Po naplánování a vytvoření trajektorie lze výsledky zobrazit v programu rviz.

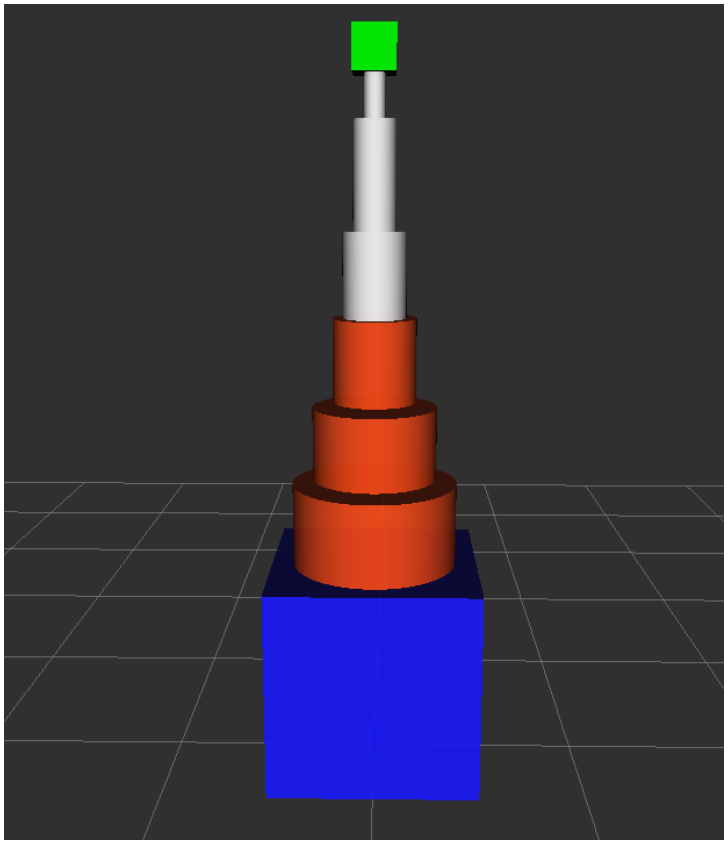
Stejně jako grafické rozhraní, i ta programová umožňují kromě jednoduchého nastavení dvou poloh robotu také nastavení omezení maximálních rychlostí a zrychlení během průchodu trajektorií. Programová rozhraní také podporují plánování trajektorií po přímkách. Tyto trajektorie se zadávají jako vektor bodů, kterými má koncový efektor projít, nebo jako počáteční a koncovou polohu se zadanou maximální vzdáleností mezi dvěma po sobě jdoucími body trajektorie. MoveIt! následně interpoluje zadané body.

4.2.1 Plánování komplexní trajektorie v programových rozhraních

Obdobně jako u grafického rozhraní je možné v těch programových přidat do plánovací scény kolizní objekty a připevňovat je ke koncovému efektoru. Při následné vizualizaci v programu rviz se vykreslí kromě samotného modelu robotu i přidané kolizní objekty. Mimo tato omezení plánované trajektorie lze zadat toleranci přesného dosažení koncové polohy plánované trasy, jak z pohledu polohy tak i natočení koncového efektoru.

5 Sestavení modelu manipulátoru

Nejprve jsem sestavil vlastní model robotu pomocí sepsání souboru URDF. Můj robot je šestiosý manipulátor se šesti planárními klouby. Pro jednoduchost jsem jako ramena mého manipulátoru zvolil základní trojrozměrná tělesa. Základna mého robotu je krychle, jeho ramena jsem zavedl jako válce. Následně jsem vytvořil ještě koncový efektor ve formě kvádru. Jako poslední krok jsem doplnil omezení na polohu, rychlost a zrychlení kloubů. Tento model jsem následně použil i pro procvičení práce s plánovacím programem MoveIt! a s programem rviz.



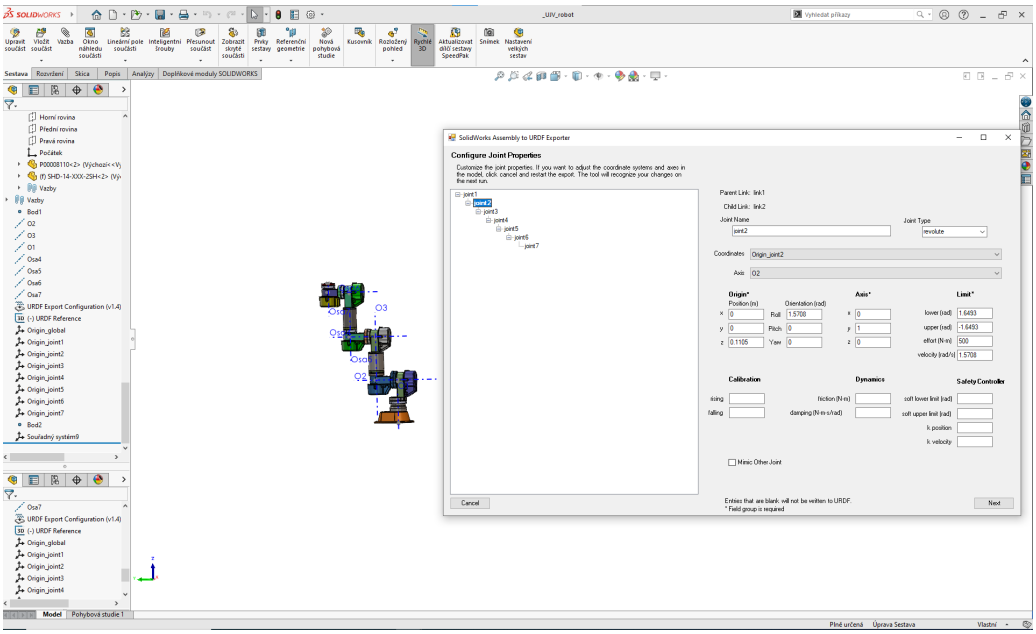
Obrázek 5.1: Model šestiosého manipulátoru.

5.1 Získání modelu sedmiosého manipulátoru z programu SolidWorks a plánování jeho trajektorie

Jako druhý model robotu jsem dostal model vytvořený v programu SolidWorks. Tento model byl vytvořen na Katedře kybernetiky Fakulty aplikovaných věd pro účely vývoje nového manipulátoru. Abych mohl využít tento model v systému ROS, musel jsem nejprve získat model ve formátu URDF. Jelikož SolidWorks neumožňuje standardně exportovat modely v tomto formátu, použil jsem rozšíření SolidWorks to URDF Exporter [10].

5.1.1 Postup Exportování modelu z programu SolidWorks

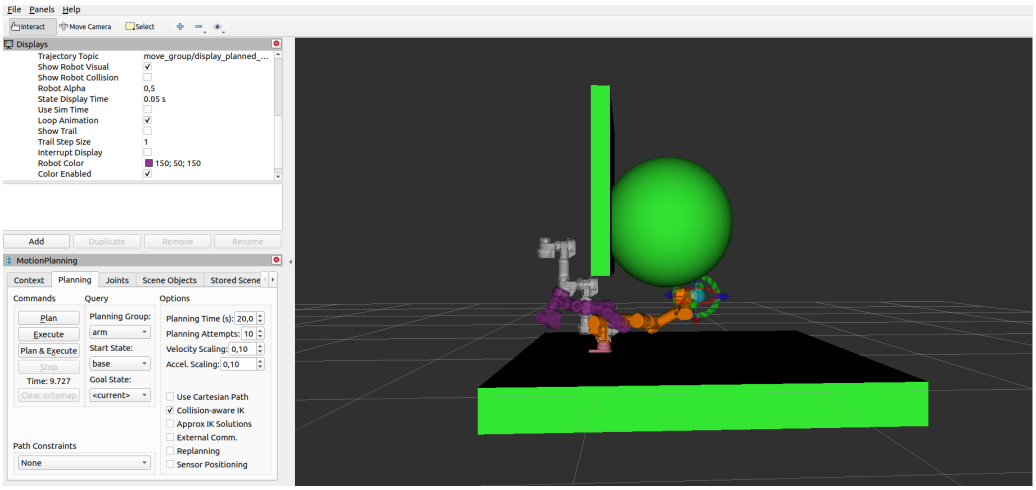
Po nainstalování rozšíření SolidWorks to URDF Exporter je nutné před jeho samotným použitím nejprve doplnit model o několik částí. Prvním krokem je vytvoření souřadného systému, který bude použit jako souřadný systém světa pro model URDF. Následně je doporučeno každému kloubu robotu přiřadit osu kolem, která bude určovat směr jeho pohybu. U prismatických kloubů se jedná o osu ,ve které se bude kloub vysouvat a u planárních je to osa rotace. Pokud obsahuje model koncový efektor s pohyblivými klouby, je nutné doplnit osy i u těchto kloubů. V případě, že tyto osy nejsou doplněny, pokusí se je Exporter vytvořit sám a mohou být nepřesné.
Po doplnění modelu o potřebné osy můžeme postoupit k využití samotného rozšíření SolidWorks to URDF Exporter. Toto rozšíření spustíme přes horní lištu programu SoliWorks, kde v záložce Nástroje najdeme možnost Tools a v ní Export as URDF. Otevře se grafické rozhraní, ve kterém se nastaví první rameno robotu a souřadný systém světa. Poté se doplní ostatní ramena do modelu, kde každému ramenu se vyplní všechny části, ze kterých se skládá a nastaví se počet ramen, která na zadané rameno navazují. Každému ramenu se také doplní kloub, kterému se zadá jeho typ a osa pohybu. V tuto chvíli je již možné model exportovat kliknutím na tlačítko Preview and Export. Po stisknutí se zobrazí okno, ve kterém je možné zkontrolovat vyplněné údaje a doplnit chybějící potřebné parametry. Každému kloubu je třeba ještě doplnit omezení maximální rychlosti, momentu a maximální a minimální polohu. Následně je možné exportovat zvolený model ve formátu URDF. Při exportování se vytvoří nová složka, která bude obsahovat nejen samotný model URDf, ale i 3D modely ramen a soubory popisující jejich grafickou podobu.



Obrázek 5.2: Exportování modelu do souboru URDF z programu SolidWorks

5.1.2 Plánování trajektorie sedmiosého manipulátoru

Abych mohl využít program MoveIt! pro plánování pohybové trajektorie mého manipulátoru, musel jsem nejprve projít konfigurací modelu v programu Setup Assistent. Po dokončení konfigurace jsem zkusil vytvořit několik pohybových trajektorií přes grafické rozhraní programu MoveIt!.
Nejprve jsem ověřil funkčnost plánování a správné nastavení modelu pomocí jednoduché trajektorie mezi dvěma náhodně zvolenými body. Po ověření, že se model chová podle očekávání, jsem přešel k plánování komplexní trajektorie, na které se manipulátor musel vyhnout dvěma kolizním objektům, aby se dostal do požadované polohy. Jako kolizní objekty jsem zvolil kvádr a kouli, které jsem umístil do trajektorie, kterou plánovač vytvořil před jejich přidáním. Aby manipulátor během svého pohybu nenarazil do podlahy, umístil jsem do plánovací scény ještě jeden kvádr, který představoval omezení pracovního prostoru ve formě podlahy.



Obrázek 5.3: Plánování trajektorie pro sedmiosý manipulátor.

Na obrázku je vidět celá plánovací scéna se třemi kolizními objekty, počáteční polohou manipulátoru (bílí model), koncovou polohou (oranžový model) a vzorovým průchodem naplánované trajektorie (fialový model).

Po naplánování této pohybové trajektorie jsem zkusil ještě aproximovat pohyb po kružnici tím, že jsem obě polohy manipulátoru nastavil blízko k různým částem kolizního objektu ve tvaru koule. Tento pokus ovšem nevedl vždy na pohyb po kružnici, občas MoveIt! vytvořil mnohem delší a komplikovanější trasu mezi zadanými polohami. Jako poslední trasu jsem vyzkoušel pohyb po přímce, jelikož u mého ručně psaného modelu šestiosého manipulátoru toto zadání velmi často nevedlo k nalezení řešení. Jak se ukázalo u přesnějšího modelu, nebyl s pohybem po přímce žádný problém a chyby s hledáním podobné trajektorie u mého šestiosého modelu byly zřejmě způsobeny nedokonalostí architektury samotného manipulátoru, kde hledaná trajektorie často procházela singularitami, nebo vytvářela kolize ramen.

6 Možnosti přenosu získané trajektorie do externího systému

Po sestavení modelu manipulátoru a naplánování jeho pohybové trajektorie bylo mým dalším úkolem zjistit možnosti přenosu získané trajektorie, a případně i dalších dat, do externího řídícího systému. Tento externí systém byl reprezentován řídícím systémem Rexygen[7] od firmy REX Controls s.r.o.

6.1 Exportování naplánované trajektorie

Při řízení robotů pomocí systému Rexygen[7] se pohybová trajektorie importuje jako soubor jednotlivých bodů trajektorie ve formátu csv, který je možné načítat blokem RM_Feed [8]. Abych tedy mohl využít pohybovou trajektorii vytvořenou pomocí systému ROS, musel jsem nejprve tuto trajektorii uložit ve vhodném formátu. Protože programy ROS Node umožňují mimo využívání knihoven systému ROS také všechny standardní funkce jazyka, ve kterém jsou psány, není žádný problém v uložení dat do vybraného souboru.
Vytvořil jsem program TrajSaver.cpp, který po spuštění odebírá data z prostředí /move_group/display_planned_path, kam naplánovanou pohybovou trajektorii nahrává program MoveIt! Získaná data rozdělí na jednotlivé body trajektorie a následně z každého bodu získá všechny kloubové souřadnice, jejich rychlosti a zrychlení. Zprávy, ve kterých jsou data posílaná přes prostředí, se skládají ze základních datových typů, v případě plánované trajektorie z čísel s pohyblivou desetinnou tečkou. Tyto hodnoty si následně uloží do pole, ze kterého je vypíše na konzoli a zároveň je uloží do zvoleného souboru. Jelikož systém Rexygen používá jako základní úhlové jednotky stupně, na rozdíl od systému ROS, který používá radiány, musel jsem hodnoty trajektorie převést na správné jednotky. Po načtení a následném vypsání naplánované pohybové trajektorie se data uloží do souboru *trajektorie.csv*. Soubor používá jako oddělující znak středník a jako desetinné znaménko tečku.
Po ověření, že program plní svoji funkci, jsem doplnil výpis do dalšího souboru s názvem *trajektorie_plus_speed.csv*. Do tohoto souboru byly vepsány všechny polohy trajektorie, ale obsahoval upravené hodnoty úhlových

rychlostí. Aby bylo možné použít naplánovanou trajektorii v systému Rexygen, který běží s pevnou periodou, využil jsem pro aproximování úhlové rychlosti diferenci dvou po sobě jdoucích poloh vydělenou periodou určenou pro programy v Rexygenu. Místo všech hodnot úhlových zrychlení vypíše program nuly.

$$\hat{v_j} = (P_{i,j} - P_{i-1,j})/T \tag{6.1}$$

Kde $\hat{v_j}$ značí hledaný odhad rychlosti pro kloub j, $P_{i,j} - P_{i-1,j}$ je rozdíl dvou po sobě jdoucích poloh ve stupních a T je perioda Rexygenu.

6.2 Další možnost exportování dat

Stejným způsobem jako u naplánované trajektorie lze exportovat do souboru libovolná data ze systému ROS. Využít této možnosti by se dalo například pro získání dat pro simulace pomocí programu Gazebo nebo pro získání vzorových dat ze senzorů a kamer. Získané hodnoty se nemusí pouze ukládat, ale můžeme využít síťové komunikace, kterou oba systémy podporují. Tímto způsobem by bylo možné nahrávat data a parametry v reálném čase přímo do řídícího algoritmu a zpět. Vzájemné propojení systémů by mohlo být přínosné při kontrole funkčnosti navrženého řídícího algoritmu pomocí metody SIL, neboli Software in the loop. Tato metoda spočívá v připojení řídícího algoritmu k programově simulovanému modelu řízeného systému, kde řídící systém Rexygen by přijímal simulované chování řízeného systému z prostředí ROS.

7 Diskuze o dalších možnostech práce

Tato kapitola rozebírá další možné směry využití systému ROS ve spojení s tématem této práce a naznačuje možnosti, jak na tuto práci navázat.

7.1 Další možnosti plánování pohybových trajektorií

Jedním z dalších možných způsobů pro využití programu MoveIt! v úloze plánování pohybových trajektorií je možnost snímání trajektorie pomocí senzorů z reálného robotu nebo jeho simulace v programu Gazebo. Při tomto snímání je potřeba, aby měl robot ve svých kloubech senzory schopné zaznamenávat jejich polohy a fyzicky pohybovat s rameny a koncovým efektorem, zatím co MoveIt! zaznamenává kloubové souřadnice této trajektorie.
Samozřejmě není nutné pro plánování trajektorie využít program MoveIt!. Pokud vypočteme inverzní kinematickou úlohu, můžeme sestavit vlastní algoritmus pro určení pohybové trajektorie a výslednou trajektorii uložit pro aplikování v externím systému.

7.2 Nástupce ROS systém ROS2

Systém ROS podle všech známých informací končí svůj vývoj s verzí Noetic Ninjemys, který by měl mít podporu do roku 2025. Hlavními důvody pro ukončení vývoje je zastaralost systému a množství duplicitních knihoven, které vytvořili vývojáři systému, kterými jsou dobrovolníci z celého světa. Nástupcem systému ROS se pomalu stává systém ROS2[4]. Tento systém zachovává stejnou architekturu jako jeho předchůdce a slibuje rozvoj v oblasti moderních technologií. Systém ROS2 je již nyní dostupný ve verzích Galactic a Foxy. S novým systémem ROS2 byly vyvinuty i nové knihovny a programy včetně nového rviz2 a MoveIt!2. Tento systém je možné plně používat i na platformě MS Windows 10 a novějších verzích operačních systémů firmy Microsoft.
Na tuto práci by bylo možné navázat například zpracováním zadání pro systém ROS2. Očekávám, že postup by byl velmi podobný, jelikož ROS2 funguje téměř stejně jako ROS a programy používané pro práci s roboty

jsou jen modernizované, ale obsahují stejné funkce. Mohlo by být zajímavé porovnat naplánované trajektorie a zkoumat jejich rozdíly, zda například došlo k vylepšení plánovacích algoritmů.

7.3 Systém ROS a počítačové vidění

Systém ROS umožňuje používání knihoven počítačového vidění jako možnost pro řízení robotů pomocí kamerového snímání okolního prostředí. Využívá především knihovnu OpenCV [1], konkrétně verzi OpenCV5. Tato knihovna obsahuje nástroje pro komplexní analýzu obrazu a získání všech informací o okolí robotu pro potřeby plánování trajektorií.

8 Závěr

Hlavním cílem této práce bylo prozkoumat možnosti modelování robotů v systému ROS. Určit způsob plánování pohybových trajektorií a následně exportovat vytvořené trajektorie do externího řídícího systému. V první části práce jsem popsal všechny potřebné základní vlastnosti a funkcionality systému ROS. Zmínil jsem sestavovací program Catkin nezbytný pro práci se systémem ROS na platformě Linux Ubuntu. Rozebral jsem architekturu balíku a možnost tvorby programů v jazycích C++ a Python. Dále jsem uvedl způsob jakým spolu programy komunikují a také jádro roscore, které zajišťuje chod systému. Popsal jsem také způsoby spouštění programů.
V druhé části práce jsem se zabýval hlavními způsoby modelování robotů a plánování jejich pohybových trajektorií. Popsal jsem funkci knihovny tf, která zajišťuje práci se souřadnými systémy a jejich transformacemi. Ukázal jsem formát pro tvorbu modelů robotů URDF a jeho nejpoužívanější části. Dále jsem popisoval dva programy používané pro práci s roboty a vizualizační program rviz, který jsem použil pro vykreslování modelů. Nakonec jsem uvedl základní informace o simulačním programu Gazebo a programu na plánování trajektorií MoveIt!.
Třetí část práce je věnována plánování standardních a komplexních pohybových trajektorií v systému ROS. Pro plánování jsem používal program MoveIt! a jeho nástroje pro plánování v grafickém rozhraní pro program rviz. Poté jsem popsal možnost využití programových rozhraní v jazycích Python a C++, které umožňují využít MoveIt! v programech systému ROS a program MoveIt Commander, který představuje možnost ovládání grafického rozhraní pomocí příkazového řádku. Pro všechna rozhraní jsem uvedl možnost plánování komplexních trajektorií pomocí přidávání kolizních objektů do okolí robotů a zmínil jsem možnost připevnění objektů ke koncovému efektoru.
Ve čtvrté části jsem ukazoval tvorbu dvou modelů manipulátorů. Nejprve jsem sestavil vlastní model šestiosého manipulátoru pomocí ručního sepsání souboru URDF. Následně jsem se věnoval modelu sedmiosého manipulátoru v programu SolidWorks, který jsem musel exportovat ve vhodném formátu URDF pomocí rozšíření SolidWorks to URDF Exporter. Po získání modelu jsem provedl potřebnou konfiguraci pro program MoveIt! a naplánoval několik komplexních pohybových trajektorií.

Pátá část byla věnována možnostem exportování naplánované trajektorie a jejímu možnému využití v systéme Rexygen. Vytvořil jsem program TrajSaver, který načte vytvořenou trajektorii, vypíše ji do konzole a následně ji uloží do souboru ve formátu CSV. Zjistil jsem, že mohu plně využít pouze plánování kloubové polohy, protože v programu MoveIt! není možné zadat pevný časový rozestup mezi dvěma body trajektorie a proto nelze v systému Rexygen využívat plánované kloubové rychlosti a zrychlení. Následně jsem popsal další možnost exportování dat pomocí komunikace po síti a možnosti využití systému ROS pro kontrolu správného fungování řídícího algoritmu pomocí metody SIL.

Celkově bych jako výsledek této práce označil jako uspokojivý, jelikož se mi podařilo úspěšně exportovat naplánované pohybové trajektorie a umožnit jejich využití v externím řídícím systému. Nebylo bohužel možné využít naplánované rychlosti a zrychlení z plánované trajektorie, ale nahradil jsem chybějící informace o rychlostech pomocí odhadu hodnot přes diference. Překvapilo mne množství specializovaných nástrojů, které jsou součástí systému ROS a jejich intuitivní a relativně snadné používání, doplněné o dobře sepsanou dokumentaci s jednoduchými návody pro obsluhu systému.

Literatura

[1] Bradski, G. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*. 2000. Dostupné z: http://opencv.org/.

[2] *Gazebo docs* [online]. Open Robotics, 2021. [cit. 2022/14/5]. Gazebo Documentation. Dostupné z: https://gazebosim.org/docs.

[3] Islam, F. et al. RRT*-SMART: A Rapid Convergence Implementation of RRT*. *International Journal of Advanced Robotic Systems: Robot Motion*. 2012. Dostupné z: http://save.seecs.nust.edu.pk/pubs/ICMA2012.pdf.

[4] Macenski, S. et al. Robot Operating System 2: Design, architecture, and uses in the wild. *Science Robotics*. 2022, 7, 66, s. eabm6074. doi: 10.1126/scirobotics.abm6074. Dostupné z: https://www.science.org/doi/abs/10.1126/scirobotics.abm6074.

[5] *MoveIt ROS* [online]. PickNik Robotics, 2022. [cit. 2022/14/5]. MoveIt! Documentation. Dostupné z: https://moveit.ros.org/.

[6] Quigley, M. – Gerkey, B. – Smart, W. *Programming Robots with ROS: A Practical Introduction to the Robot Operating System*. O'Reilly Media, 2015. Dostupné z: https://books.google.cz/books?id=Hnz5CgAAQBAJ. ISBN 9781449325510.

[7] *Rexygen* [online]. Rex Control s.r.o., 2022. [cit. 2022/19/4]. Systém Rexygen. Dostupné z: https://www.rexygen.com/.

[8] *REXYGEN Documentation RM_Feed* [online]. Rex Control s.r.o., 2022. [cit. 2022/19/4]. REXYGEN Documentatin. Dostupné z: https://www.rexygen.com/doc/ENGLISH/MANUALS/BRef/RM_Feed.html#x322-32100020.

[9] *Ros wiki* [online]. Open Robotics, 2020. [cit. 2022/14/5]. Ros Documentation. Dostupné z: https://wiki.ros.org/.

[10] *SolidWorks to URDF Exporter* [online]. Stephen Brawner, 2022. [cit. 2022/14/5]. SolidWorks to URDF Exporter Documentation. Dostupné z: http://wiki.ros.org/sw_urdf_exporter.

[11] Şucan, I. A. – Moll, M. – Kavraki, L. E. The Open Motion Planning Library. *IEEE Robotics & Automation Magazine*. December , , 4, s. 72–82. doi: 10.1109/MRA.2012.2205651. Dostupné z: https://ompl.kavrakilab.org.

Seznam obrázků