VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY
DEPARTMENT OF CONTROL AND INSTRUMENTATION

PROGRAMOVÁNÍ VÝUKOVÉHO ROBOTICKÉHO MANIPULÁTORU
PROGRAMMING OF AN EDUCATIONAL ROBOTIC MANIPULATOR

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **David Lichosyt**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. Tomáš Lázna**
SUPERVISOR

BRNO 2020

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní program **Automatizační a měřicí technika**

Ústav automatizace a měřicí techniky

Student: David Lichosyt ***ID:*** 203279

Ročník: 3 ***Akademický rok:*** 2019/20

NÁZEV TÉMATU:

Programování výukového robotického manipulátoru

POKYNY PRO VYPRACOVÁNÍ:

Cílem bakalářské práce je oživení výukové robotické buňky od firmy FANUC s využitím softwarové platformy ROS a realizace demonstrační úlohy s manipulátorem.

1. Seznamte se s platformou Robot Operating System (ROS) a jejím rozšířením ROS-Industrial.
2. Ověřte možnosti programování manipulátoru FANUC pomocí existujících nástrojů na bázi ROS.
3. Implementujte zvolené řešení pro ovládání robotu.
4. Proveďte analýzu vybrané demonstrační úlohy, která využije výhod systému ROS, a navrhněte pro ni modulární řešení. Zohledněte bezpečnostní aspekty.
5. Po dohodě s vedoucím práce implementujte moduly nezbytné pro základní demonstraci prvků výukové robotické buňky.
6. Diskutujte, jakým způsobem byste řešil chybějící moduly.

DOPORUČENÁ LITERATURA:

[1] KHAN, Khasim A., Revanth R. KONDA a Ji-Chul RYU. ROS-based control for a robot manipulator with a demonstration of the ball-on-plate task. Advances in Robotics Research. 2018, 2(2), 113-127. ISSN 2287-4976.

Termín zadání: 3.2.2020 ***Termín odevzdání:*** 8.6.2020

Vedoucí práce: Ing. Tomáš Lázna

doc. Ing. Václav Jirsík, CSc.
předseda rady studijního programu

ABSTRAKT

Práce se v obecné části se práce věnuje teorii ohledně platformy ROS, její nástavbě ROS Industrial, balíčku MoveIt a zpracování obrazu. V praktické části jsou tyto znalosti jsou dále aplikovány na robotické rameno Fanuc LR Mate 200iD/4S, jeho kontrolér Fanuc 30iB Mate a průmyslovou kameru za účelem vytvoření systému pro pohyb figurek po šachovnici v duchu hry známé jako dáma.

KLÍČOVÁ SLOVA

ROS, ROS Industrial, MoveIt, robotický manipulátor, Fanuc, LR Mate 200iD/4S, OpenCV, zpracování obrazu, počítačové vidění

ABSTRACT

The thesis in theoretical part is about inner arrangement of ROS platform and principles behind ROS Industrial, Moveit and image processing. In its practical part is this theoretical knowledge applied on robotic manipulator Fanuc LR Mate 200iD/4S, its controller Fanuc 30iB Mate and industrial camera to create system for moving pieces across the board like in game name checkers.

KEYWORDS

ROS, ROS Industrial, MoveIt, robotický manipulator, Fanuc, LR Mate 200iD/4S, OpenCV, image processing, computer vision

LICHOSYT, David. *Programování výukového robotického manipulátoru.* Brno, 2020. Dostupné také z: https://www.vutbr.cz/studenti/zav-prace/detail/126934. Bakalářská práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřicí techniky. Vedoucí práce Tomáš Lázna.

PROHLÁŠENÍ AUTORA O PŮVODNOSTI DÍLA

Prohlašuji, že svou závěrečnou práci jsem vypracoval samostatně pod vedením vedoucího závěrečné práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené závěrečné práce dále prohlašuji, že v souvislosti s vytvořením této závěrečné práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

V Brně dne: 7. června 2020 ……

Podpis autora

PODĚKOVÁNÍ

Chtěl bych poděkovat svému vedoucímu práce panu Ing. Tomáši Láznovi za odbornou výpomoc při řešení úlohy, konzultace a návrhy v mnoha bodech práce, rychlou komunikaci, a hlavně ochotu pomáhat.

Obsah

Seznam obrázků

Seznam tabulek

ÚVOD

Práce pojednává o programování robotického manipulátoru značky Fanuc za pomoci platformy ROS a její nástavby ROS-I. Nejedná se o samostatný operační systém, jak by mohl napovídat název Robot Operating System, ale o určitý softwarový nástroj, který dílčí programy zabalí do většího celku a usnadňuje jejich vzájemnou komunikaci a výměnu dat. Ta probíhá na pozadí za pomocí ROS topics, services, action servers a messages, pomocí kterých probíhá přenos dat mezi jednotlivými programy, snímači, nebo i samotným robotem. Díky tomuto není potřeba se zabývat samotnou komunikací na nejnižší úrovni a je možno hned přejít na samotný vývoj programů a algoritmů pro pohyb a oživení robota. V této práci je užito verze ROS Melodic.

Práce je věnována tvorbě softwaru pro ovládání robotického ramene pro pohyb figurek ze hry dáma po šachovnici. Hlavním bodem práce je tedy úrpgra,, který by měl být schopen převzít jakoukoliv situaci rozloženou na šachovnici a následně dát uživateli možnost s tímto rozpoložením figurek pracovat. Software byl rozdělen do tří částí, a to první umožňující detekci šachovnice a figurek ze známé hry dáma, rozeznat jejich hráčskou příslušnost a hodnost, tedy zda-li se jedná o pěšce, či dámu a pozici. Druhou částí by měla být herní logika, nebo rozhraní nabízející možnost uživateli naplánovat pohyb figurky dle vlastní volby na jiné volné políčko na šachovnici z konzole programu. Program čte změny situace na šachovnici v reálném čase, tudíž by teoreticky bylo možné, aby hráč pohyboval s figurkami ručně, ale jelikož musí být klec při provozu manipulátoru uzavřena, toto řešení není doporučeno. Třetím z programů bude skript zpracovávající informace z uživatelského rozhraní, čímž vytvoří plán pohybu a předává jej dále do specializovaných balíčkům MoveIt a ROS industrial, které vykonají pohyb na reálném rameni. Program je dále psán tak, aby bylo možno na práci navazovat a implementovat například herní umělou inteligenci, grafické uživatelské rozhraní apod.

Struktura je nejprve rozdělena do teoretického rozboru, který je rozdělen na dvě kapitoly. První z nich pojednává o samotné platformě ROS, její nástavbě ROS Industrial, doplňujících balíčků MoveIt a dostává se přes aplikaci inverzní kinematické úlohy až k popisu užitého manipulátoru a jeho kontroléru. Druhá část čtenáři přibližuje dění za oponou zpracování obrazu a popisu, jak probíhá detekce jednotlivých geometrických tvarů. Následně je popsán hrubý teoretický postup návrhu řešení, který v další kapitole přechází v realizaci, zde jsou podrobně popsány funkce jednotlivých podprogramů celistvého softwaru a způsob komunikace mezi nimi. Poslední kapitolou je rozbor chybějících modulů, nedostatků práce a návrhy řešení pro jejich opravu.

1 PLATFORMA ROS A MANIPULÁTORY

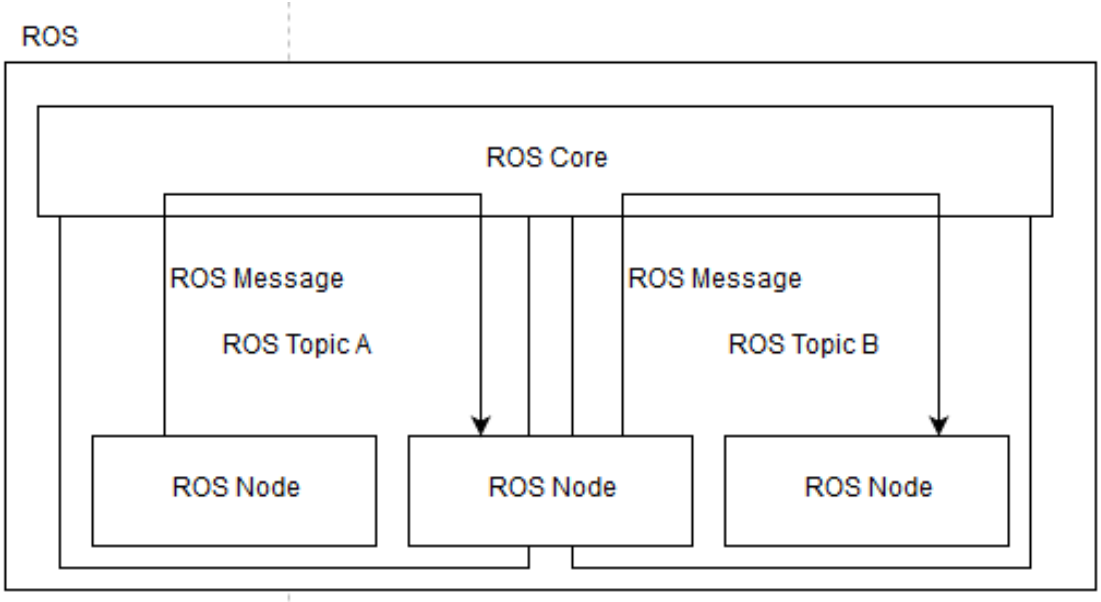
Robot Operating System je open source framework poskytující knihovny a nástroje pro vývojáře aplikací pro ovládání robotických zařízení. Mimo jiné zpřístupňuje nástroje jako hardwarovou abstrakci, vizualizaci, knihovny, drivery pro jednotlivá zařízení, komunikaci jak uvnitř systému, tak i vně a mnoho dalšího.

Jednou z velikých výhod frameworku ROS je nezávislý na programovacím jazyce, lze v něm prozatím programovat v C++, pythonu, nebo Lispu, pro Javu a Lua jsou experimentální knihovny ve vývoji. Dokonce je možné psát každý podprogram v jiném jazyce. Je také nanejvýš vhodný pro systémy s dlouhou délkou trvání programů (až nepřetržitý provoz).

Provozování několika procesů na bázi ROS jsou reprezentovány v architektuře *graph* na principu peer-to-peer sítě, kde se procesy označují jako *nodes*, které můžou přijímat a vysílat data, stavy, plánování a další zprávy. Navzdory významu rychlé odezvy v robotice, ROS sám o sobě nepracuje v reálném čase. Veškeré informace v této kapitole pochází ze zdrojů [1] [2] [3], [4], [5], [6], [8] a [9].

1.1 Základní konstrukce platformy ROS

Jak již bylo zmíněno v úvodu, ROS je určený k propojování dílčích programů neboli ROS nodů, do většího celku. Celý ROS systém je tedy rozdělen do menších kategorií, přičemž v textu dále jsou zmíněné čtyři nejhlavnější z nich, ty můžeme vidět na obrázku Obrázek 1.1.



Obrázek 1.1: Příklad základní konstrukce ROS systému. Zdroj [1]

Jednotlivé ROS systémy jsou schopny společně komunikovat za pomocí TCP/IP protokolu skrze síť, a to ať už jde o další ROS server spuštěný na počítači, či jiném terminálu nebo o robotický manipulátor sdělující informace o svém stavu a čekající na následující instrukce.

1.1.1 ROS Core

Pod tímto pojmem je možno si představit centrum celého dění, veškerá komunikace mezi nody prochází právě skrze ROS Core. Aby jednotlivé nody mezi sebou komunikaci vůbec navázaly, je potřeba, toto „jádro“ spustit jako první. Je spustitelné příkazem:

```
$ roscore
```

ROS Core se automaticky spustí i v případě, pokud doposud spuštěn nebyl, voláním programu příkazem:

```
$ roslaunch <adresář> <launch_soubor>
```

Jeho spuštěním je vytvořen TCP server, jenž otevře předem definovaný port na adrese localhost (127.0.0.1). Tím je umožněna komunikace přes rozhraní ethernet s dalšími počítači, nebo zařízeními kompatibilními s ROSem.

1.1.2 ROS Node

Node je program, který provádí nějaké operace, výpočty, nebo zpracování dat v systému ROS, je tedy do něj přidat klientskou knihovnu ROSu. Jednotlivé nody jsou společně zkombinovány do diagramu (graph), ve kterém mezi sebou mohou komunikovat a předávat si data za pomoci ROS topiců. Tento diagram se dá zobrazit příkazem:

```
$ rosrun rqt_graph rqt_graph
```

Výsledek tohoto příkazu můžeme vidět na obrázku Obrázek 1.2, kde jsou aktivní dva nody, a to jeden publisher a jeden subscriber, a jeden topic. Publisher node s názvem */publisher* vysílá zprávy do topicu */my_topic*, zatímco subscriber jménem */listener* zprávy ze stejného topicu přijímá.

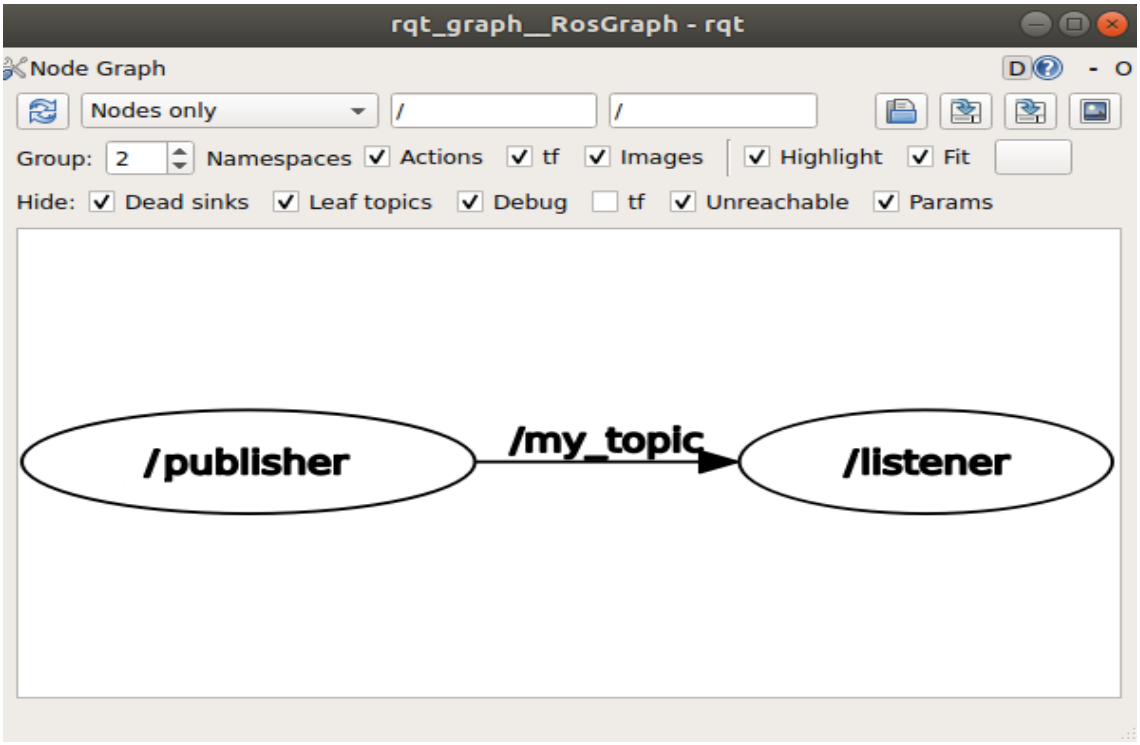
Běžný systém na ovládání manipulátoru většinou obsahuje mnoho nodů, přičemž každý má nějakou svou specifickou úlohu. Tyto jednotlivé nody lze zobrazit v konzoli příkazem:

```
$ rosnode list
```

Výhodami tohoto rozdělení na dílčí části je ku příkladu zvýšená tolerance chyb, neboť pád jednoho nodu nezapříčiní pád celého systému, nebo možnost programování jednotlivých nodů v různých jazycích. Toho je docíleno díky tomu,

že podrobnosti o jejich implementaci jsou dosti skryté a diagramu odhalují minimální informace o svém API.

Všechny spuštěné nody mají jedinečný název (graph resource name), kterým se rozeznávají zbytkem systému. Například */robot_state_publisher* je node zpracovávající data aktuálních pozic jointů manipulátoru.



Obrázek 1.2 Ukázka graph diagramu. Zdroj [1]

1.1.3 ROS Topic

Pro přiblížení pojmu topic si lze představit „kanál“, po kterém si jednotlivé nody posílají informace pomocí messagů. Node zveřejňující data do topicu se nazývá publisher a node odebírající data z topicu se označuje jako subscriber. Komunikace probíhá jednosměrným proudem a subscriber data přijme ihned po jejich uveřejnění publisherem, kde jsou zpracovány callback funkcí.

Nody samy o sobě neví, komu data posílají, ani od koho je přijímají, stačí, že ví, které topicy poslouchat a po kterých informace sdílet. Ke každému topicu může být přiřazeno více publisherů i subscriberů. Seznam topiců je možno zobrazit příkazem:

```
$ rostopic list
```

Jednotlivá témata se dále dají poslouchat, tedy vypisovat v konzoli, příkazem:

```
$ rostopic echo /topic_name
```

Kde „/topic_name“ je název existujícího topicu nalezeného v topic listu.

1.1.4 ROS Message

Zpráva neboli message, je jednoduchá struktura dat posílaná mezi nody skrze topic. Může být ve formě jednoduchého datového typu (například integer, floating point, string, apod.), nebo dokonce pole, či struktury. Message mají tedy striktně definovaný formát a často mohou obsahovat i informaci o času odeslání ve své hlavičce.

1.2 Catkin

Catkin je oficiální build systém ROSu a nástupce originálního systému rosbuild. Cílem bylo vytvořit konvenčnější systém s větší kompatibilitou a lepší přenosností. K zajištění fungovaní na úrovni CMake pracovních postupů, využívá kombinaci CMake maker a Python skriptů. V důsledku toho jsou postupy Catkinu hodně podobné jako postupy CMaku, ale navíc skýtá podporu pro automatické vyhledávání packagů a sestavování několika na sobě závislých projektů v jednom okamžiku.

Zajišťuje tedy generaci knihoven, spustitelných souborů, skriptů apod. ze zdrojového kódu, tento vygenerovaný výsledek označujeme jako *target*. Zdrojový kód je dále organizován do packagů, které mohou obsahovat jeden a více targetů.

Pro použití catkinu k vytvoření targetů je potřeba vytvořit správný workplace, dále stačí jen aplikovat příkaz:

```
$ catkin build
```

Tento příkaz vyhledá všechny package v pracovním adresáři a ze zdrojových kódů udělá targety. V našem případě to budou převážně spustitelné soubory.

1.3 RVIZ

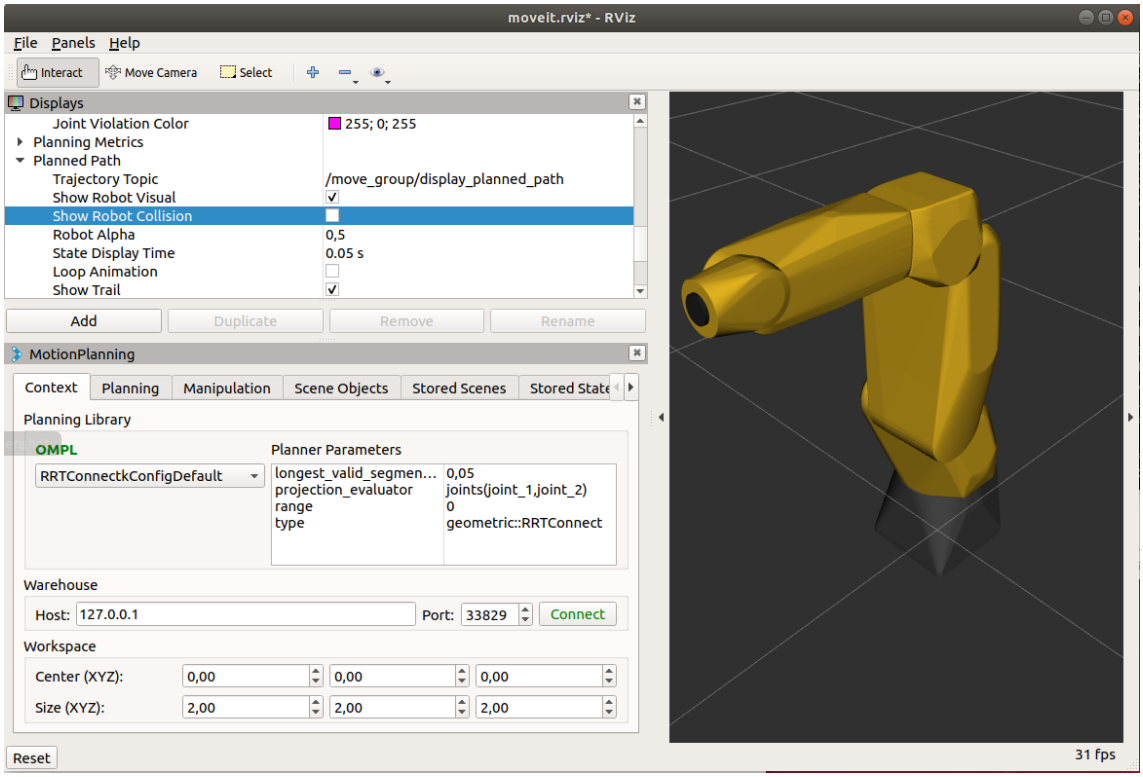
RVIZ je nástroj určen pro 3D vizualizaci ROS messagů. Umožňuje simulovat a zobrazovat roboty, prostředí, ve kterém pracují a data ze snímačů, například pointcloudy (technika vykreslování pomocí bodů), obrazy počítačového vidění apod. Za předpokladu, že nám na pozadí běží roscore, lze jej spustit příkazem:

```
$ rosrun rviz rviz
```

Na obrázku **Chyba! Nenalezen zdroj odkazů.** lze vidět konkrétní rameno F anuc LR Mate 200iD vizualizované pomocí nástroje RVIZ. Tohoto zobrazení se dá dosáhnout příkazem:

```
$ roslaunch fanuc_lrmate200id_moveit_config demo.launch
```

U tohoto příkazu už není potřeba mít spuštěný roscore, neboť začíná klíčovým slovem roslaunch. Příkaz nespouští přímo RVIZ, ale Fanuc MoveIt package, který pak následně sám RVIZ spustí a pošle mu nutná data o modelu ramene, o základní pozici jointů apod.



Obrázek 1.3 Vizualizace ramena Fanuc LR Mate 200iD/4S v nástroji RVIZ.

1.4 Nástavba ROS Industrial

ROS-I je taktéž open source projekt rozšiřující dosavadní možnosti platformy ROS pro robotiku a průmyslovou automatizaci a je k ní zpětně plně kompatibilní.

Implementace je možná dvěma způsoby a to instalací ROS-I na počítač za pomoci příkazu:

```
$ sudo apt-get install ros-melodic-industrial-core
```

Alternativou je implementace package *industrial_core* přímo do pracovního adresáře příkazem:

```
$ git clone -b indigo-devel https://github.com/ros-industrial/industrial_core.git
```

Tato nástavba se dále zaměřuje se na kvalitu kódu a jeho spolehlivost. Obsahuje proto automatické postupy hodnocení kódu a posuzování jeho úrovně, které dále sděluje uživateli. Hlavním cílem této nástavby je však sjednocení formátu dat, jejich tvorby a zpracování pro všechny roboty.

ROS Industrial přidává do ROSu interface libraries (ovladače), kde jsou k dispozici modely průmyslových manipulátorů, gripperů, sensorů apod, ale hlavním bodem jsou určitě „Unified Robot Description Formats“ (URDFs), což jsou soubory nesoucí informace v podobě XML specifikací například o modelu robota, senzorech, jointech apod. Každá tato specifikace se dále odpovídajícím způsobem překládá do jednoho, či více programovacích jazyků (např. C++, python), kde je lze dále zpracovávat a na jejich základech vyhodnocovat akce.

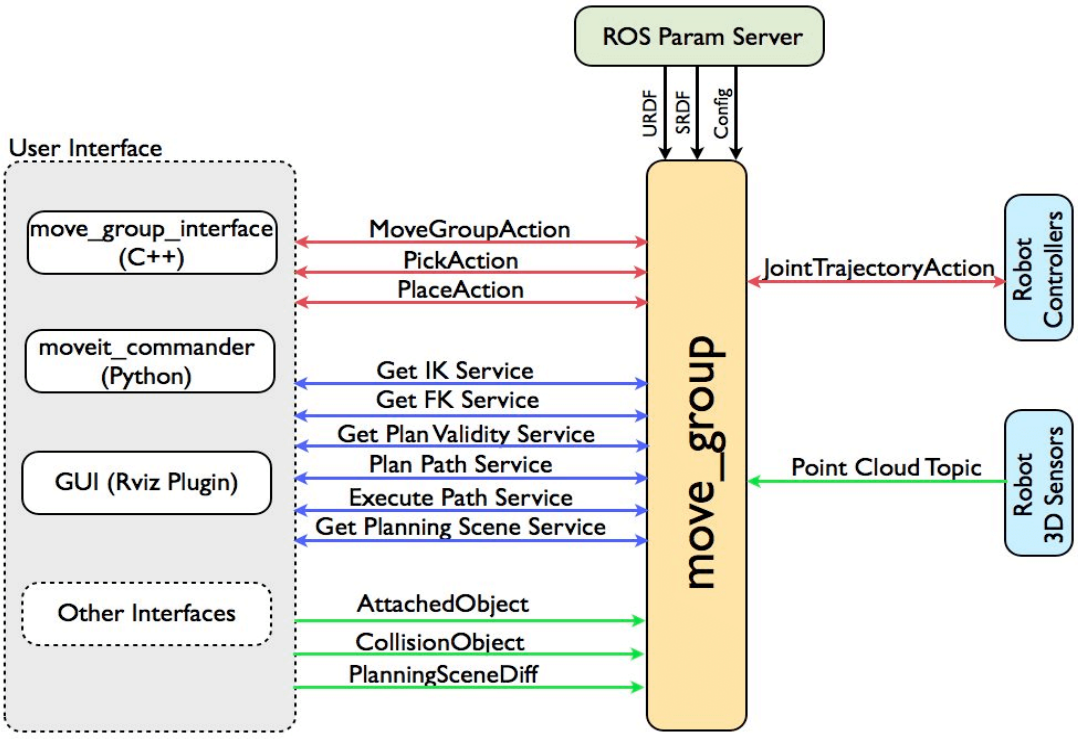
1.5 MoveIt!

MoveIt je rozšíření ROSu fungující na nejvyšší úrovni. Řadí se mezi další open-source programy, který zajišťuje komunikaci mezi ramenem a ROSem, ovladače pro širokou škálu robotů a manipulátorů apod. Také nabízí možnost plánování pohybu a kontrolu kolizí, dokáže simulovat pohyb manipulátoru, nebo i zobrazovat pohyby reálného ramena v reálném čase pomocí RVIZu.

Pro jeho užití je důležité jej nejprve nainstalovat, a to verzi pro ROS Melodic, toho lze jednoduše docílit příkazem:

```
$ sudo apt-get install ros-melodic-moveit
```

Hlavním nodem celého MoveItu je *move_group*, který dohromady spojuje individuální části programu a umožňuje uživateli snadnější ovládání skrze uživatelské rozhraní. Na obrázku Obrázek 1.4 je vyobrazena svrchní úroveň MoveIt vnitřní architektury.



Obrázek 1.4 Vnitřní architektura MoveItu. Zdroj [6]

1.5.1 Fanuc MoveIt! Drivery

Základem komunikace jsou package *fanuc_driver* a *fanuc_experimental.* Na jejich úspěšné sestavení catkinem je zapotřebí mít nainstalovaný ROS Industrial package za pomocí příkazu:

```
$ sudo apt-get install ros-melodic-industrial-core
```

Další alternativou je implementace package *industrial_core* do pracovního adresáře příkazem:

```
$ git clone -b indigo-devel https://github.com/ros-industrial/industrial_core.git
```

Skrze něj komunikují Fanuc ovladače s kontrolérem manipulátoru, který podporuje programovací prostředí KAREL. Oba Fanuc balíčky obsahují značné množství spustitelných *launch* souborů pro jednotlivé modely manipulátorů. Tyto soubory se následně využívají pro spouštění simulací v RVIZu, samotné komunikace s reálným ramenem, spouštění *Parametr serverů*, nebo MoveIt Assistanta pro generaci URDF a SRDF souborů.

1.5.2 MoveIt! Setup Assistant

MoveIt Setup Assistant je grafické uživatelské rozhraní sloužící k vytváření, nebo konfiguraci robotů a manipulátorů. V této práci je následně použit pouze pro vygenerování URDF a SRDF souborů. Nejprve je potřeba Assistanta spustit příkazem:

```
$ roslaunch moveit_setup_assistant setup_assistant.launch
```

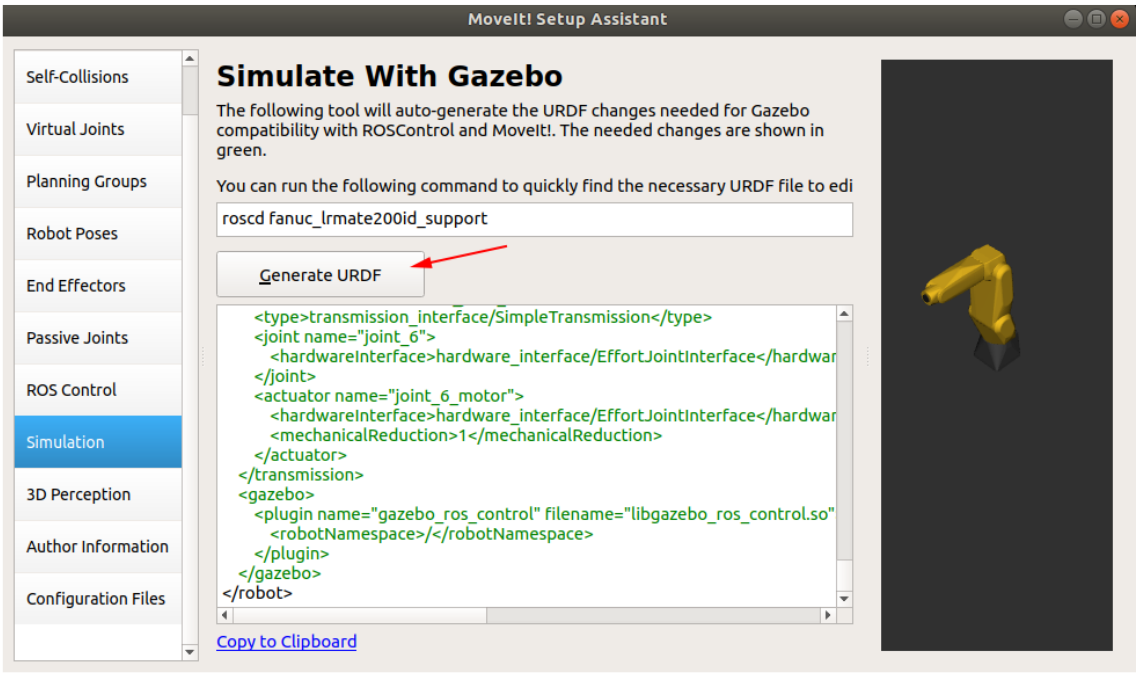
Následně je potřeba zvolit možnost editace existujícího ramene a vyhledat konfigurační adresář fanuc_lrmate200id_moveit_config. Pro náš konkrétní případ existuje pohodlnější varianta, a to v podobě příkazu:

```
$ roslaunch fanuc_lrmate200id_moveit_config setup_assistant.launch
```

Tím se dostaneme do základního okna tohoto průvodce. Je automatický nastavený tak, aby editoval konfiguraci už našeho existujícího ramena, tudíž stačí zmáčknout tlačítko v pravém dolním rohu s nápisem „Load Files“. Učiněním tohoto kroku se nám otevře široká škála možností editaci virtuálního modulu ramene, například hranice kolizí, výchozí pozice manipulátoru, přídavné nástavce na konci manipulátoru, informace o jointech apod. Jelikož je ale vytvořený už pro naše konkrétní rameno, je tudíž kompletní a není potřeba nic z toho editovat.

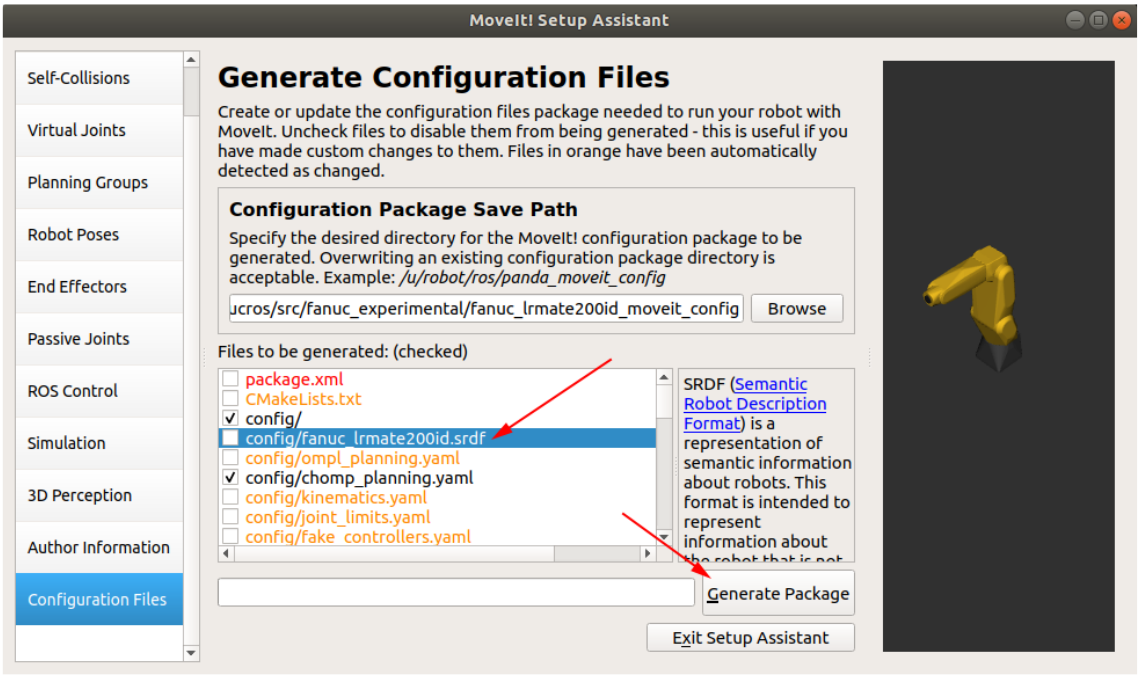
Naším cílem bude vytvoření URDF a SRDF souborů, které jsou potřebné pro konečné řízení našeho robotického ramene. Nejprve si ukážeme generaci souboru URDF, která lze provézt v záložce „Simulation“. Jak můžeme vidět na obrázku 2.2,

zde se stiskem tlačítka „Generate URDF“ zobrazí kód, který lze zkopírovat a uložit do prázdného dokumentu s příponou urdf.



Obrázek 1.5 Generace URDF souboru.

Na obrázku 2.3 je vyobrazena generace SRDF souboru. Je potřeba přepnout na záložku „Configuration Files“, dále zaškrtnout políčko u položky „config/fanuc_lrmate200id.srdf“ a nakonec stisknout tlačítko „Generate Package“. Tím se požadovaný SRDF soubor vygeneruje.



Obrázek 1.6 Generace SRDF souboru.

V případě tohoto manipulátoru není potřeba SRDF soubor nikterak generovat, neboť je již součástí package *fanuc_experimental*. [7]

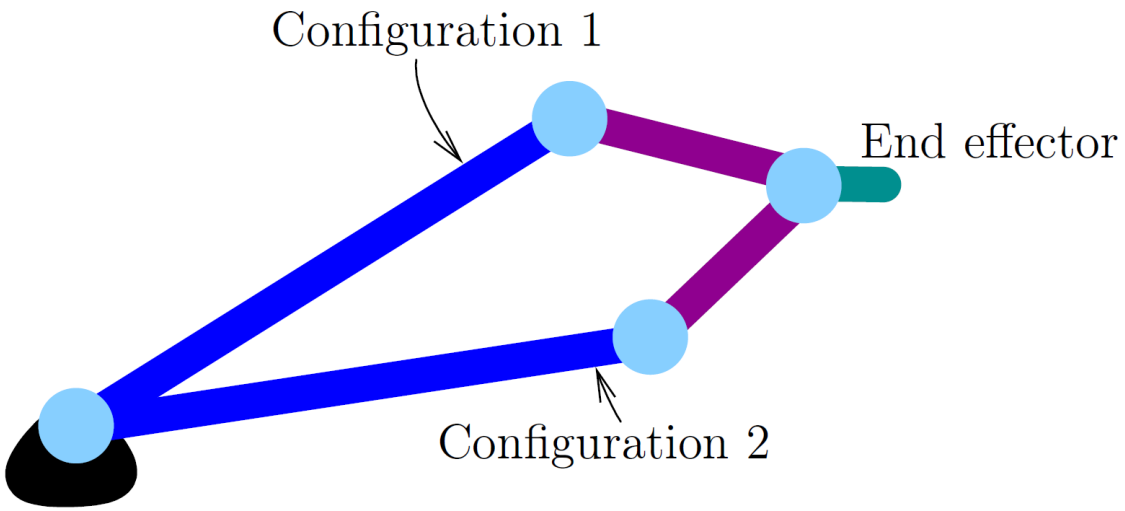
1.6 Manipulátory

Pojem „robotické rameno“ se začal vyskytovat od pozdních sedmdesátých let dvacátého století a později si našel uplatnění v mnoha oborech. Největšími z nich jsou vesmírné projekty a automatizační průmysl. Ačkoliv původní využití byla především na manipulaci s radioaktivními, nebo bio-hazardními materiály, později s klesajícími náklady na jejich výrobu a použití se začala používat na mnohé pro člověka obtížné úkoly za účelem odstranění lidských chyb a snížením výdajů při automatizované průmyslové výrobě.

Samotné rameno bývá složeno z více segmentů (anglicky označovaných link), které jsou spojeny pomocí kloubů (anglicky joint). Ovládáno je z příslušného kontroléru, ke kterému je připojeno a z hlediska bezpečnosti má vymezený prostor pro pohyb, kde člověk z pravidla za provozu nemá přístup. V Této podkapitole bylo užito zdroje [10].

1.6.1 Inverzní kinematická úloha

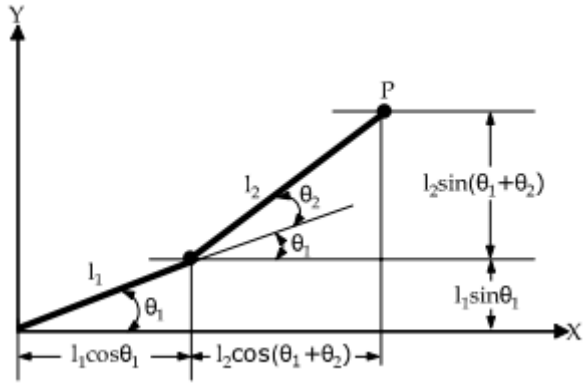
Ačkoliv inverzní kinematická úloha je řešená v balíčku MoveIt, v rámci robotických manipulátorů stojí za zmínku. Je opakem výpočetního procesu přímé kinematické úlohy, která popisuje výpočet polohy end-effectoru v kartézských souřadnicích v závislosti na úhlech natočení kloubů modelu manipulátoru. Inverzní kinematika tedy počítá hodnoty polohy jointů z požadované výsledné pozice gripperu v kartézských souřadnicích a úhlu jeho natočení. Tady ale nastává problém, kdy s rostoucím počtem kloubu přibývá počet možných variací jejich stavů, jak lze vidět na obrázku Obrázek 1.7.



Obrázek 1.7 Ukázka více řešení inverzní kinematické úlohy. Zdroj [23]

Řešení inverzní kinematické úlohy bývá časově náročné, zvláště pro manipulátory pracující v reálném čase. K dispozici jsou dvě metody řešení, a to geometrická a algebraická.

První z možných postupů je aplikovatelný na jednodušší manipulátory, jako například manipulátor se dvěma stupni volnosti. Tato metoda spočívá v rozložení geometrie manipulátoru na částečné jednodušší geometrické úlohy, jak lze vidět na obrázku Obrázek 1.8. Jelikož se práce konkrétně této problematice nevěnuje, není potřeba uvést detailní popis výpočtu.



Obrázek 1.8 Geometrické řešení inverzní kinematické funkce. Zdroj [24]

Algebraická metoda je na druhou stranu používaná převážně pro složitější manipulátory, kde se geometrické řešení stává problematickým. Pro výpočet manipulátoru, na příklad se šesti stupni volnosti, bude užito rovnice (1.1), kde T představuje transformační matici, r rotační elementy (jeho indexy i a j = 1, 2, 3), p značí souřadnice polohového vektoru a q je hodnota natočení kloubu.

$$ {}^{0}_{6}T = \begin{bmatrix} r_{11} & r_{12} & r_{13} & p_x \\ r_{21} & r_{22} & r_{23} & p_y \\ r_{31} & r_{32} & r_{33} & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = {}^{0}_{1}T(q_1)\,{}^{1}_{2}T(q_2)\,{}^{2}_{3}T(q_3)\,{}^{3}_{4}T(q_4)\,{}^{4}_{5}T(q_5)\,{}^{5}_{6}T(q_6) \qquad (1.1) $$

Nejprve je potřeba rovnici vynásobit inverzní maticí ${}^{0}_{1}T(q_1)$, tím na pravé straně rovnice dojde k vynásobení této původní matice s maticí jí inverzní a vznikne jednotková matice I. Rovnice dostane tedy nový tvar (1.2).

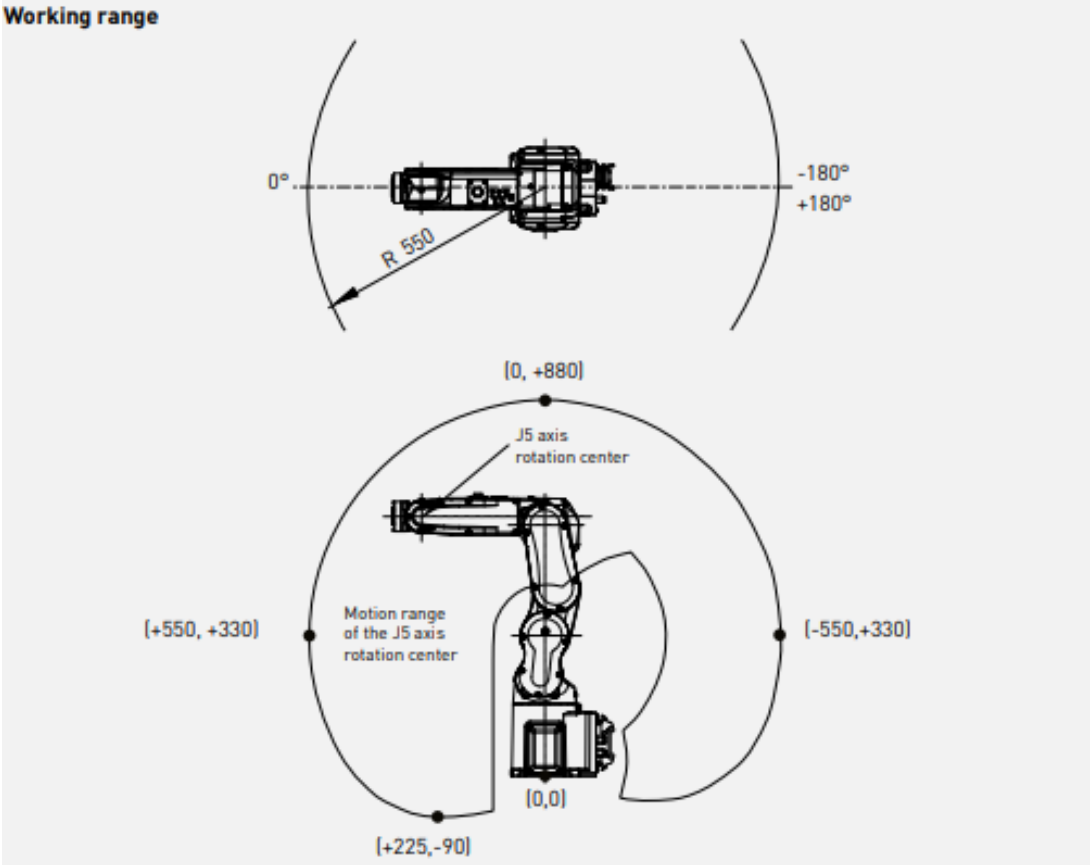
$$ [{}^{0}_{1}T(q_1)]^{-1}\,{}^{0}_{6}T = {}^{1}_{2}T(q_2)\,{}^{2}_{3}T(q_3)\,{}^{3}_{4}T(q_4)\,{}^{4}_{5}T(q_5)\,{}^{5}_{6}T(q_6) \qquad (1.2) $$

Obdobným způsobem se vyjádří i další rovnice. Řešení lze následně nalézt ve tvaru 12ti nelineárních rovnic, kde jedinou neznámou proměnnou levé strany bude q_1. Dvanáct nelineárních prvků bude poté rovno nule, nebo budou představovat

konstanty. Následně je možné řešit jednotlivé prvky q v libovolném pořadí. Podkapitola čerpá ze zdrojů [23] a [24].

1.6.2 Manipulátor Fanuc LR Mate 200iD/4S

V případě této práce byl zvolen kompaktní šestiosý manipulátor japonské značky Fanuc LR Mate 200iD/4S s nosností až 4 kg a dosahem 550 mm, který je ovládán pomocí kontroléru R-30iB Mate. Robotické rameno disponuje ochranou proti vniknutí pevných těles a vody úrovně IP67.



Obrázek 1.9 Pracovní dosah manipulátoru Fanuc LR Mate 200iD/4S. [10]

Na obrázku Obrázek 1.9 je znázorněn pracovní prostor zvoleného manipulátoru za předpokladu, že není omezen například bezpečnostní klecí.

1.6.3 Kontrolér Fanuc R-30iB Mate

Kontrolér Fanuc R-30iB Mate je ideální volbou pro menší robotická ramena a byl navržen speciálně pro roboty řady LR-Mate.

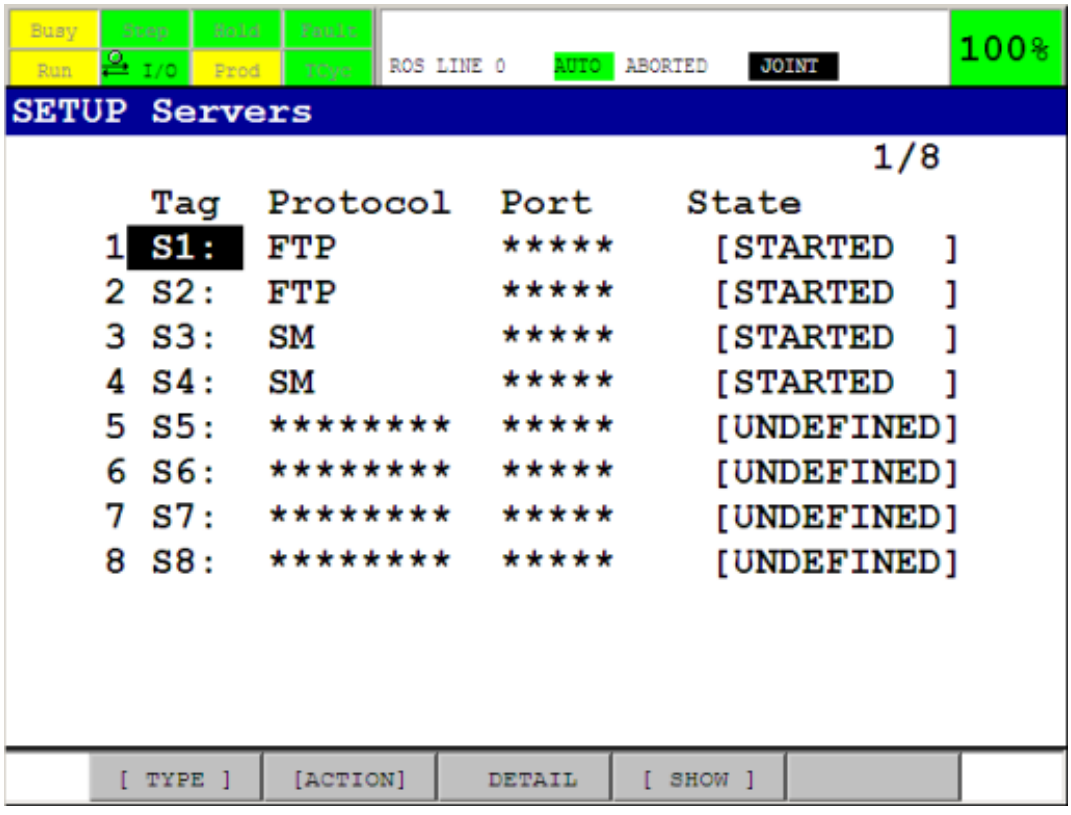
Tento kontrolér je vybaven dotykovým ovladačem FANUC iPendant-Touch užitý pro ovládání robotického ramene, programování v jazyce KAREL, nastavování registrů, spouštění programů a mnoho dalších aktivit.

User socket messaging

Důležitým předpokladem pro funkčnost je takzvaný „User Socket Messaging“, který zajišťuje komunikaci mezi vnitřním systémem kontroleru robotického ramena a vzdáleným počítačem za pomoci rozhraní ethernet a TCP/IP protokolu. Diky tomu dokáže kontrolér vysílat jednotlivé ROS Messages počítači a také přijímat zprávy informující o následném dění. Bez tohoto modulu se s ramenem pomocí ROSu komunikovat nedá. Pokud tedy v paměti kontroléru chybí, je nutné jej dokoupit a nainstalovat.

Instalace ROS Industrial ovladače na kontrolér

Pro zajištění komunikace pomocí ROSu, je důležité, aby byl nejen nainstalovaný na počítači, ale i na samotném kontroléru. Možnostmi jsou stažení ovladače *fanuc_driver*, nebo *fanuc_driver_exp*. Druhý zmíněný je schopen pracovat na vyšší frekvenci, má flexibilnější kód, lépe pracuje na síti a mnohem méně zatěžuje hardware, proto byl také zvolen. V první řadě je tedy potřeba stáhnout *fanuc_driver_exp* ze sítě GitHub [12] a jeho extrahované soubory přenést do root adresáře flash disku. Následně se flash disk zapojí do kontroléru Fanuc R-30iB Mate a pomocí teach pendantu se nainstalují spuštěním souborů *uninstall.cm* a *install.cm*. Tímto je instalace dokončena a můžeme se přesunout na konfiguraci.



Obrázek 1.10 Nastavení server tagů. Zdroj [11]

Programy *ros_state* a *ros_traj* potřebují pro svou funkci dva volné *Server Tagy*. Je tedy potřeba otevřít obrazovku nastavení *Host Comm*, kam je možné se dostat možnostmi „Menu -> SETUP -> Host Comm“ na tech pendantu a následně „SHOW -> Servers“. Obvykle bývají tagy 1 a 2 použity již zabudovaným FTP serverem, proto se většinou pro ROS používají tagy 3 a 4, jak lze vidět na obrázku Obrázek 1.10. Tyto tagy je potřeba nastavit na protokol *SM* a inactivity timeout 1 minuta, komentář je volitelný. Po jejich nastavení musí mít status *STARTED*, pokud tomu tak není, stačí daný tag označit a zvolit nejprve „ACTION -> DEFINE“ a následně „ACTION -> START“.

Dalším krokem bude rezervování flagů, integer registrů a pozičních registrů, přičemž poziční registry potřebují dva hned po sobě jdoucí, například 1 a 2. V případě této práce byly zvoleny volné tagy 4 a 5, integer registry 21 až 25, poziční registry 1 a 2 a nakonec flagy taktéž 1 a 2. Tyto informace se musí zapsat do konfigurace programů *ros_state* a *ros_traj*. Tuto konfiguraci lze otevřít stisknutím tlačítka „Select“ na teach pendantu, dále „TYPE -> KAREL Progs“, poté je potřeba zvolit *ros_state* nebo *ros_traj*, stiskem tlačítka „ENTER“ potvrdit, následně tlačítko „DATA“ a nakonec „TYPE -> KAREL Vars“ celou sekvenci navigace je nutno zakončit stiskem klávesy „ENTER“. Nastavení těchto dvou programů můžeme vidět v následujících tabulkách Tabulka 1.1 a Tabulka 1.2.

Tabulka 1.1 Konfigurace programu ros_traj.

Název	Typ	Hodnota	Jednotka	Popis
checked	BOOL	True	-	Konfigurace zkontrolována
f_msm_drdy	INTEGER	2	-	Flag pro signál *motion sm data ready*
f_msm_rdy	INTEGER	1	-	Flag pro signál *motion sm ready*
loop_hz	INTEGER	42	Hz	Rychlost obnovení hlavní smyčky
move_cnt	INTEGER	50	%	Hodnota CNT použita pro každou instrukci pohybu
pr_move	INTEGER	1	-	Poziční registr pro *next traj pt*
r_move_cnt	INTEGER	22	-	Int registr pro *traj segment CNT*
r_skip_sig	INTEGER	25	-	Int registr sledován SKIP CONDITIONem v Teach Pendantu
r_tseg_vel	INTEGER	21	-	Int registr pro *traj segment joint velocity percentage*
s_acpt_dlay	INTEGER	1000	ms	Prodleva návratu na *sock_accept*
s_tag_nr	INTEGER	5	-	Číslo použitého server TAGu
s_tcp_nr	INTEGER	11000	-	Přiřazení TCP portu
tp_val_trstp	INTEGER	1	-	Hodnota na nastavení *r_skip_sig* registru pro signál *traj stop*
trjpt_buf_sz	INTEGER	6	-	počet míst *traj pts* v bufferu
um_clear	BOOL	True	-	Vymazání user menu při startu

Tabulka 1.2 Konfigurace programu ros_state.

Název	Typ	Hodnota	Jednotka	Popis
checked	BOOL	True	-	Konfigurace zkontrolována
loop_hz	INTEGER	42	Hz	Rychlost obnovení hlavní smyčky
s_accpt_dlay	INTEGER	1000	ms	Prodleva návratu na *sock_accept*
s_tag_nr	INTEGER	4	-	Číslo použitého server TAGu
s_tcp_nr	INTEGER	11002	-	Přiřazení TCP portu
um_clear	BOOL	True	-	Vymazání user menu při startu

Následujícím krokem je úprava programu *ros_movesm*, kde je potřeba zapsat hodnoty použitých flagů a registrů.

Během pokusu a spuštění ROSu na rameni nastaly potíže a to, že programy *ros_state* a *ros_traj* nešly zapnout najednou, protože jeden z nich se vždy vypnul. Toto bylo vyřešeno editací volajícího programu *ros*, kde byly první tři řádky programu změněny následovně:

```
1: RUN ROS_STATE
2: RUN ROS_MOVESM
3: CALL ROS_RELAY
```

2 ZPRACOVÁNÍ OBRAZU

Při zpracování obrazu je důležité nejprve detekovat čtvercovou šachovnici a na ni poté kulaté figurky. Aby k tomuto rozpoznávání mohlo dojít, je nejprve potřeba obraz patřičně upravit. Úpravy začínají u rozmazání, či jiné filtrace nežádoucích chyb obrazu, pokračují přes filtraci skrze určité barevné spektrum, kde mimo jiné rozlišujeme RGB a HSV formát, až po samotnou detekci hran a tvarů. Užité zdroje: [16], [17] a [18].

2.1 Konvoluce ve zpracování obrazu

Konvoluce je matematický nástroj v našem případě využitý k úpravě snímků. Principiálně se hodnota každého pixelu upravuje na základě jeho lokálního okolí.

$$a = \frac{m - 1}{2} \tag{2.1}$$

$$b = \frac{n - 1}{2} \tag{2.2}$$

Za předpokladu, že platí rovnice 2.1 a 2.2 a že m a n jsou lichá celá čísla, je konvoluce snímku s maticí známé jako kernel (značíme w) velikosti $m \cdot n$ se snímkem f(x, y) vyjádřena vztahem 2.3.

$$(w \star f)(x, y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s, t) f(x - s, y - t) \tag{2.3}$$

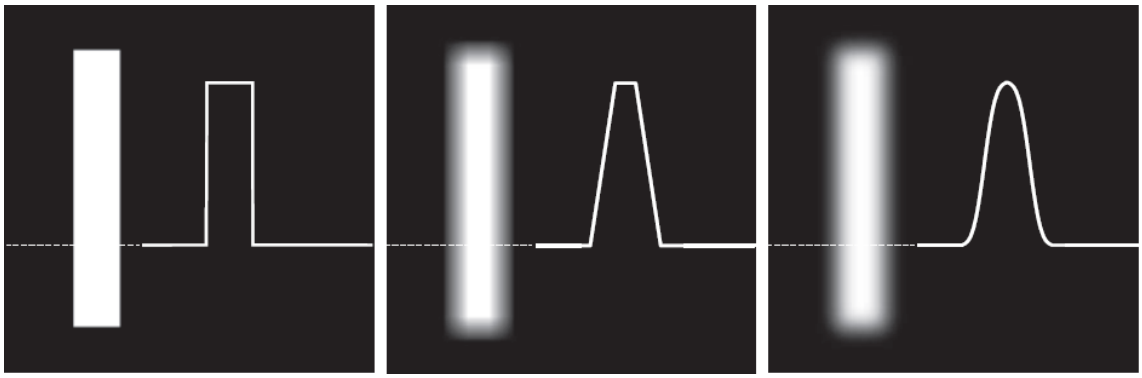
Konvoluční filtry (kernely) se dělí na filtry typu dolní propust používané pro odstranění nečistot, chyb a slabých hran před jejich detekcí, a na filtry typu horní propust, které se používají pro detekci těchto hran.

2.1.1 Konvoluční filtry typu dolní propust

Mezi kernely typu dolní propusti spadá například Gaussovo rozostření, které je popsáno dále, nebo průměrové rozostření, které každému prvku v okolí přiřadí stejnou váhu, sečte je, a nakonec podělí počtem těchto prvků.

Gaussovo rozostření, anglicky Gaussian blur, je aplikace konvoluce snímku s kernelem, jehož matice je složena z elementů určených Gaussovou funkcí. Na rozdíl od průměrového kernelu, dosahuje tato metoda rozostření značně lepších výsledků, jak lze vidět na obrázku Obrázek 2.1. Jelikož se jedná o kernel kruhového

charakteru s větším důrazem na lokální okolí, dochází k menšímu rozostření hran a také toto rozostření není závislé na orientaci objektu.



Obrázek 2.1 Srovnání původního snímku, konvoluce s průměrovým a snímku s Gaussovým kernelem. Zdroj [16]

Gaussova funkce je definována vztahem 2.4, kde σ značí směrodatnou odchylku a s a t představují souřadnice.

$$G(s,t) = \frac{1}{2\pi\sigma^2} e^{-\frac{s^2+t^2}{2\sigma^2}} \tag{2.4}$$

2.1.2 Konvoluční filtry typu horní propust

Konvoluční filtry s charakterem horní propusti jsou používány převážně k detekci hran. Jelikož je obraz chápán jako 2D diskrétní signál, je možné jej derivovat. Gradient obrázku *f* na souřadnicích (x, y) je definován jako sloupcový vektor, jak lze vidět v rovnici 2.5. Velikost vektoru ∇f značená M(x, y) je dále daná vztahem 2.6.

$$\nabla f = grad(f) = \begin{bmatrix} g_x \\ g_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} \tag{2.5}$$

$$M(x,y) = \|\nabla f\| = mag(\nabla f) = \sqrt{g_x^2 + g_y^2} \tag{2.6}$$

Kde M je hodnota velikosti změny ve směru gradientového vektoru na souřadnicích (x, y). Nyní je potřeba definovat diskrétní aproximace rovnicím 2.5 a 2.6 a následně sestavit odpovídající kernel. Pro zjednodušení použijeme kernel o rozměru 3x3 a s označením příslušných polí, jak je vyobrazeno v tabulce Tabulka 2.1 3x3 Kernel s popisem polí. Zdroj [16].

Tabulka 2.1 3x3 Kernel s popisem polí. Zdroj [16]

z_1	z_2	z_3
z_4	z_5	z_6
z_7	z_8	z_9

Nejjednodušší první derivace jsou $g_x = (z_8 - z_5)$ a $g_y = (z_6 - z_5)$, v takovém případě by gradient obrázku vypadal jako v rovnici 2.7.

$$M(x, y) = [(z_8 - z_5)^2 + (z_6 - z_5)^2] \tag{2.7}$$

Používaným kernelem realizující derivaci snímku je Sobelův operátor (Tabulka 2.2), u kterého lze vidět, že dává větší váhu středu, čímž jsou potlačeny nežádoucí malé gradienty.

Tabulka 2.2 Kernel s hodnotami pro Sobelův operátor (vlevo g_x, vpravo g_y). Zdroj [16]

1	0	-1
2	0	-2
1	0	-1

1	2	1
0	0	0
-1	-2	-1

Tyto filtry jsou následně konvolucí aplikovány na obraz, z čehož vzniknou dva různé obrazy. V jednom z nich budou zvýrazněné svislé hrany a v druhém vodorovné. Výpočtem velikosti gradientu (vzorec 2.6) vznikne výsledný obraz se zvýrazněnými hranami.

2.2 Houghova transformace pro detekci kruhů

OpenCV nevyužívá normální Houghovu transformaci, ale upravenou metodu zvanou Houghova gradientní metoda. Vstupním obrazem tohoto algoritmu je obraz detekovaných hran například po aplikaci konvoluce s filtrem dle Sobelova operátoru.

Nejprve je pro každý nenulový bod ve vstupním obraze je uvažován lokální gradient. Použitím tohoto gradientu je každý bod podél detekované hrany pod určitým sklonem navýšen v akumulátoru. Zároveň je pozice každého z těchto nenulových pixelů ve vstupním obraze zaznamenána. Pozice možných středů jsou následně vybrány z těchto bodů uložených ve dvojrozměrném akumulátoru, které

ovšem musí být nad zadaným prahem (anglicky threshold) a také musí být větší než nejbližší sousední kandidáti. Tyto možné středy jsou dále seřazeny v sestupném pořadí dle jejich akumulátorových hodnot tak, že středy s největším počtem podpůrných pixelů vyjdou napovrch. Nyní pro každý střed jsou brány v potaz všechny pixely, které byly umístěny do akumulátoru na začátku. Tyto pixely jsou seřazeny dle jejich vzdáleností od středu. Pracuje se od nejmenšího zadaného poloměru až po nejvyšší. Střed se ponechá, pokud má dostatečnou podporu okolních nenulových pixelů ve zvoleném rozsahu poloměru a pokud přesahuje minimální vzdálenost od jakéhokoliv dříve detekovaného středu. Tento algoritmus je mnohem rychlejší než běžná Houghova transformace pro detekci kruhů, a hlavně obchází použití trojrozměrného akumulátoru, který by vedl k velkému šumu při detekci. [18]

2.3 Barevná filtrace

Aby bylo možné detekovat barvy přítomné na snímku, je potřeba nejprve vytvořit takzvané masky, které definují rozptyl, ve kterém se tyto barvy nacházejí, přičemž je lze popsat dvěma způsoby, a to buď ve formátu RGB, nebo HSV. Následně se provede logický součin každého pixelu s danou maskou, kde na výstupu zůstanou pouze pixely, které do tohoto předdefinovaného rozsahu zapadaly.

Výsledný obraz lze ještě upravit na binární, což znamená, že všechny prvky dosahující určité intenzity se změní na hodnotu 255 (tedy bílou barvu) a všechny ostatní na hodnotu 0 (tedy barvu černou).

Obrázek 2.2 Příklad morfologické transformace v režimu otevírání. Zdroj [17]

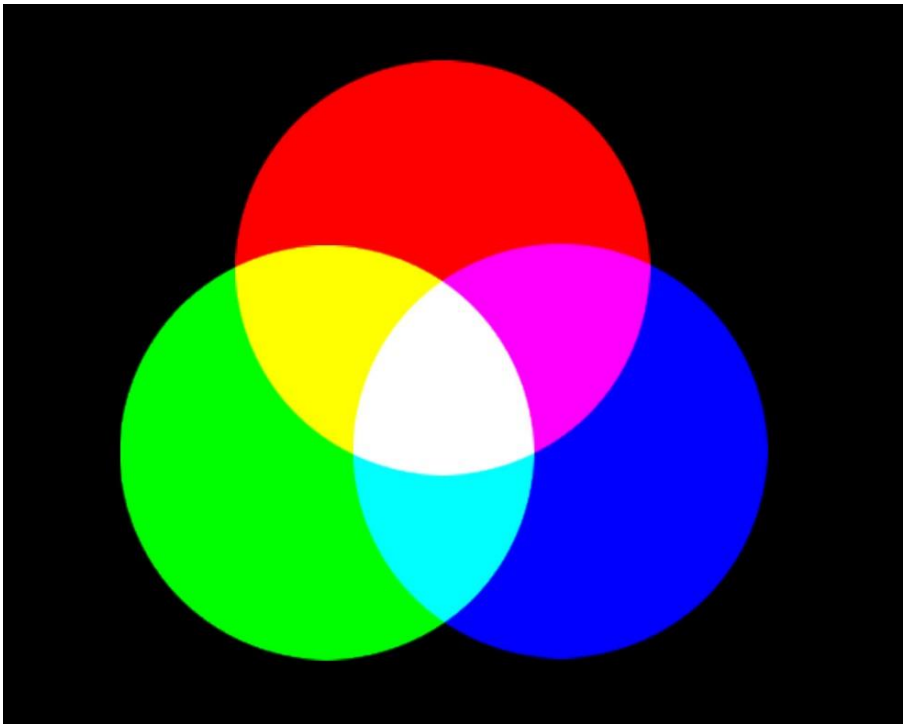
Pro finální odstranění chyb, šumu, děr a špatné filtrace barev je možno snímek dále doladit za pomoci morfologické transformace. Zde rozeznáváme dvě základní operace, tedy eroze a dilatace. Z nich lze dále nakombinovat například funkce otevření (Obrázek 2.2), která nejprve provede erozi a následně dilataci, čímž je nejprve detekovaný objekt zmenšen a následně zvětšen. Tímto postupem je odstraněn přebytečný šum okolo detekovaného objektu. Druhou takovou funkcí je

opačná funkce zavření, kde proběhne nejprve dilatace a následně až po ní eroze (Obrázek 2.3). Důsledkem toho dojde k zacelení děr uvnitř objektu.

Obrázek 2.3 Příklad morfologické transformace v režimu zavírání. Zdroj [17]

2.3.1 RGB barevný model

Zkratka RGB definuje barevný model často používaný na obrazovkách a jiných zobrazovacích zařízeních, kde je výsledná barva definována jednotlivými složkami červené, zelené a modré barvy.

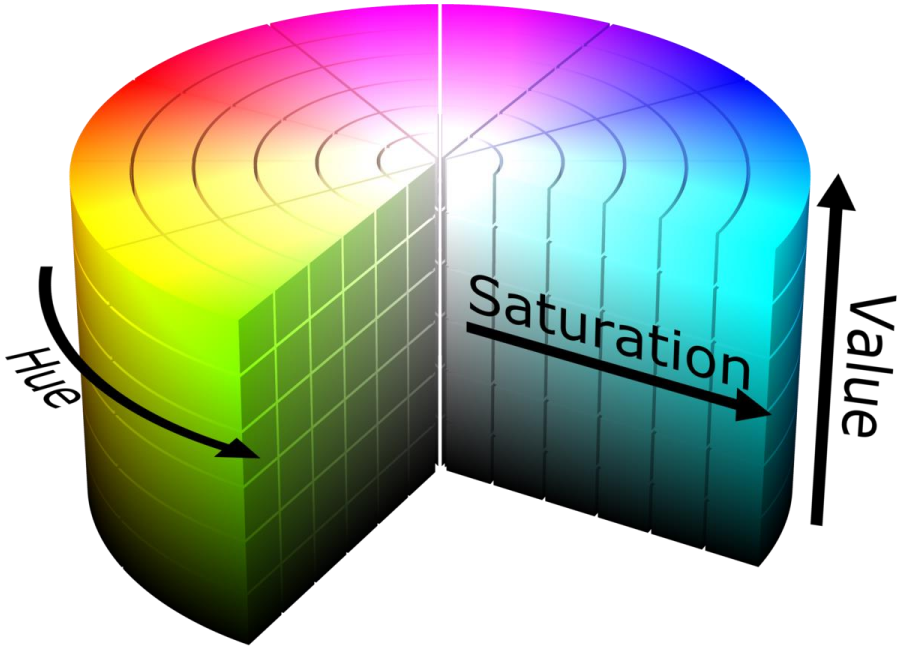


Obrázek 2.4 RGB barevný model. Zdroj [19]

Každá tato barevná složka je často definována hodnotou 0 až 255, z čehož vyplývá, že barevných kombinací je až 16 581 375. Pro detekci určité barvy o různé saturaci a jasu není tato metoda vhodná, neboť je potřeba předdefinovat několik masek, aby tento rozsah dostatečně popsaly.

2.3.2 HSV barevný model

Na rozdíl od RGB modelu HSV model popisuje barvu dle tří kritérií. Těmi jsou hue, neboli odstín, kde úhel natočení na barevném kruhu na obrázku Obrázek 2.5 definuje o jakou barvu se jedná. Dalším kritériem je saturace, tedy jak je vybraná barva sytá, hodnota je udávaná v procentech v rozsahu 0 až 100. Posledním z nich je value, česky hodnota nebo světlost, která definuje množství bílého světla obsaženého v barvě.



Obrázek 2.5 HSV barevný model. Zdroj [20]

Tento barevný model je pro počítačové zpracování barev mnohem vhodnější než dříve zmíněný RGB model, jelikož můžeme přesně definovat jednu masku obsahující jednu barvu a zbylé dva atributy se již doladí dle potřeb. Snad jedinou výjimkou je červená barva, která je jak na začátku, tak na konci barevného kola, její rozsah musí být tedy definován dvěma maskami z obou stran počátku.

2.4 OpenCV

OpenCV (Open Source Computer Vision Library) je open source knihovna pro zpracování počítačového vidění a strojové učení.

Knihovna obsahuje přes 2500 optimalizovaných algoritmu určených k detekování a rozpoznávání obličejů, objektů, geometrických útvarů, lidských aktivit nebo sledování objektů v průběhu videa a mnoho dalšího. Je často

využívána ve velkých společnostech, výzkumných organizacích, nebo vládních institucích.

OpenCV aktuálně nabízí programování v C++, Pythonu, Jave a MatLabu a podporuje operační systémy Windows, Linux, Android a Mac OS. Uplatňuje se převážně při vidění a detekci v reálném čase. V této podkapitole bylo užito zdroje [21].

2.5 Kamera

Pro získávání kvalitních snímků byla zvolena průmyslová kamera ImagingSource DFK 41BU02 s rozlišením 1280x960 px a frekvencí snímání 7,5 snímků za vteřinu. Komunikace s PC probíhá skrze rozhraní USB 2.0. Tato podkapitola čerpá ze zdrojů [13], [14] a [15].

Obrázek 2.6 Kamera ImagingSource s objektivem computar.

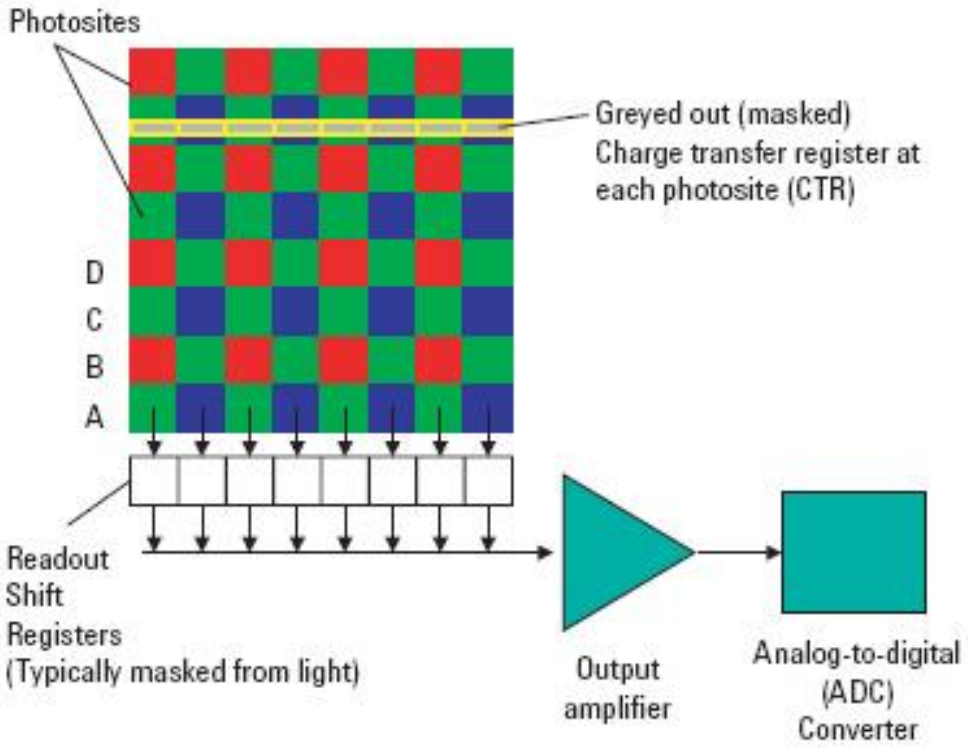
Kameru je možno napájet napětím v rozsahu 4,5 – 5,5 V stejnosměrného napětí. Přibližný proud při 5 V odpovídá 500 mA. Je vybavena čipem Sony ICX205AK typu CCD.

Ke kameře byl zvolen objektiv computar H0514-MP2 s manuálním ostřením a nastavováním jasu. Díky svým parametrům vyhovuje úloze a je schopnen zobrazit celou aktivní plochu vyhrazenou pro robotický manipulátor.

2.5.1 Porovnání CCD a CMOS čipů

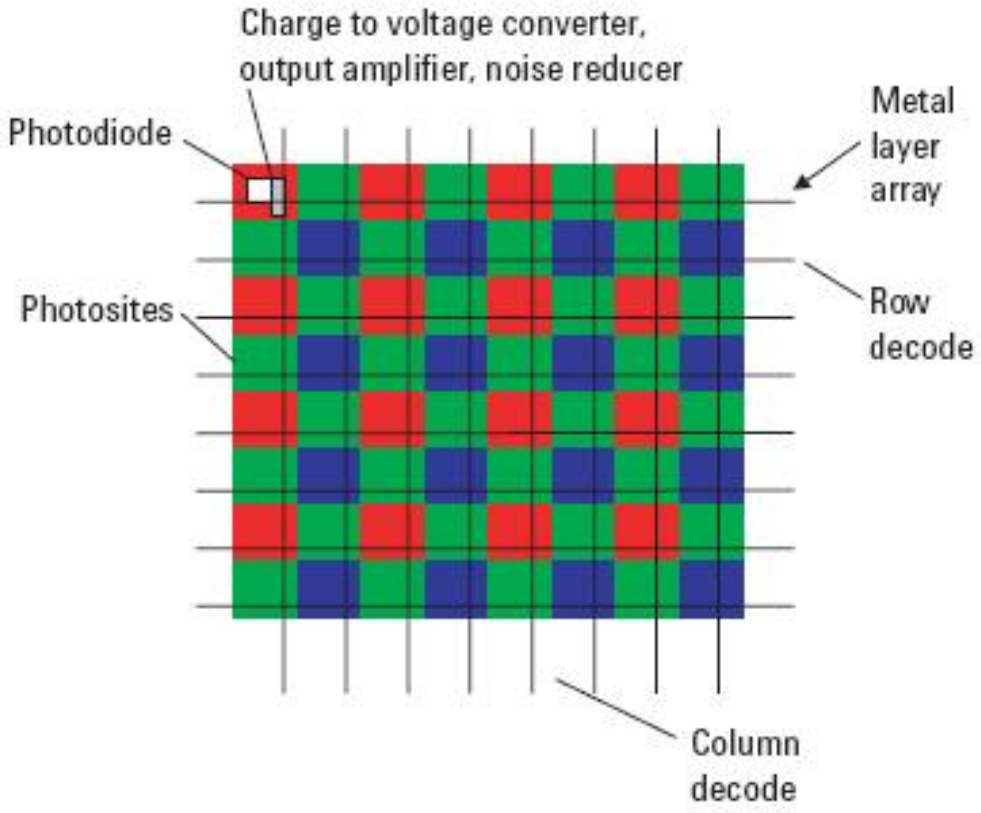
CCD snímací čip se skládá ze světlocitlivých buněk produkujících elektrický náboj jako reakci na dopadající světlo. Čtení dat z čipu probíhá po řádcích, každý z nich se nejprve načte do posuvného registru, který navazuje na zesilovač, ze kterého se data dostanou až do A/D převodníku, na jehož výstupu jsou již reprezentována digitální formou.

Tato metoda čtení je pomalá a neumožňuje načíst jen výřez obrázku, je potřeba načíst celý snímek. Na druhou stranu je kvantová efektivita EQ, tedy číslo popisující, jaká část světla je převedena na náboj, může u určitých vlnových délek dosahovat až 90%, častější ale bývají hodnoty kolem 60%, což je stále poměrně dobré ve srovnání s čipy typu CMOS, kde se tato hodnota pohybuje kolem 25%. Obraz je také v lepší kvalitě a s nižším šumem.



Obrázek 2.7 Způsob čtení dat z CCD čipů. Zdroj [14]

U CMOS čipu jsou pro zachycení pixelů použity dvojice unipolárních tranzistorů společně s křemíkovou fotodiodou. Díky tomu mají až desetkrát nižší spotřebu než dříve zmíněné CCD čipy. Jsou také levnější a rychlejší na výrobu, neboť výrobní technologický proces je obdobný jako u procesorů. Dělí se na aktivní a pasivní, přičemž aktivní obsahují navíc zesilovač a obvody na potlačení šumu.



Obrázek 2.8 Způsob čtení dat z CMOS čipů. Zdroj [14]

Souhrn rozdílů, výhod a nevýhod mezi CCD a CMOS čipy lze vidět v tabulce Tabulka 2.3.

Tabulka 2.3 Tabulka porovnání CCD a CMOS. Zdroj [14]

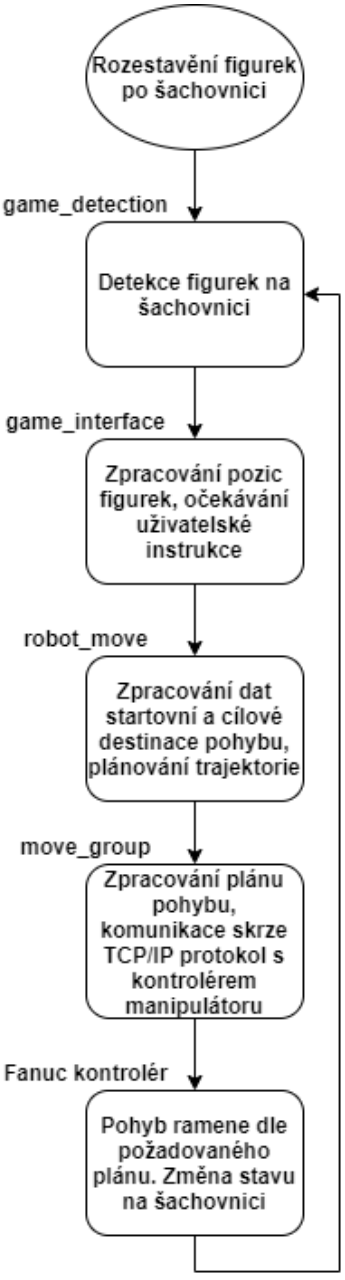
Rozdíly mezi CCD a CMOS		
Snímací čip	**CCD**	**CMOS**
Cena	vysoká	nízká
Rozměry řešení	vyšší	nízké
Spotřeba	vysoká	nízká
Kvalita obrazu	vysoká	nižší až nízká
Rozlišení	vysoké	střední
Komplexnost čipu	vysoká	nižší až nízká
Fill faktor (činná plocha)	vysoký	nízký až střední
Digitální šum	nízký	vysoký
Rychlost	nižší až vysoká	vysoká
Dynamický rozsah	vysoký	nižší
Možnost výřezu	nativně žádná	ano

Z tohoto porovnání lze usoudit, že pro zvolenou úlohu je možno použít i kameru s CMOS čipem, ačkoli s CCD bude dosaženo lepších výsledků.

3 NÁVRH ŘEŠENÍ

Smyslem zvolené úlohy je, že uživatel rozestaví po šachovnici figurky dle vlastní potřeby, proběhne detekce a uživatel z konzole počítače může figurkám naplánovat pohyb a následně jej provézt. Pozice figurek jsou snímány před každým pohybem, aby bylo možné později implementovat hru hráče s manipulátorem přímo na šachovnici.

Celý návrh algoritmu byl rozdělen do čtyř nodů, které jsou k vidění na obrázku Obrázek 3.1, poslední částí vývojového diagramu je kontrolér ovládající manipulátor, který přijímá data z počítače o pohybu robotického ramene.



Obrázek 3.1 Diagram smyčky programu.

3.1 Návrh rozložení nodů

Navrhovaný software byl rozdělen do tří dílčích programů. Prvním z nich je detekce herní plochy (node */game_detection*), kde dochází k detekci šachovnice a figurek. Druhým je samotná herní logika, nebo uživatelské rozhraní, kde by se měla řešit pravidla tahů, nebo se jen zpracovávat zásah uživatele. Posledním z programů by měl zpracovávat data o přemisťování figurek, vypočítat plán pohybu a ten poslat balíčku MoveIt, který již zajistí pohyb na reálnem manipulátoru. Všechny tyto části jsou detailněji popsány v kapitole 4.2.

3.1.1 Detekce herní plochy

Vše začíná v nodu */game_detection*, kde probíhá zpracování obrazu. Nejprve je potřeba najít šachovnici a rozdělit ji na jednotlivá políčka. Dále dojde k detekci figurek a výpočtu jejich pozic. Následně je potřeba rozlišit, kterému z hráčů figurka přísluší a o jakou figurku se jedná, tedy zdali je to dáma, či pěšec.

Data o pozicích figurek, jejich hráčské příslušnosti a úrovni (zda-li jsou hodnosti pěch, nebo dáma) se dále posílají do nodu */game_interface*.

3.1.2 Herní rozhraní

V této časti programu po příchodu informací se čeká na uživatelský zásah, tedy o zadání souřadnic XY pozice figurky, se kterou chceme pohnout a pozice místa, kam ji chceme položit. Samozřejmě jsou ošetřeny uživatelské chyby, jako určení souřadnic mimo rozsah (1 až 8), vybrání startovní pozice, která figurku neobsahuje, nebo právě naopak zvolení destinace již obsahující jinou figurku.

Tento program dále posílá data o startovní a cílové pozici do nodu */robot_move*.

3.1.3 Plánování pohybu manipulátoru

V této části řešení dochází k převedení souřadnic políčka na kartézský souřadnicový systém manipulátoru, který má počátek v jeho základně.

Po vypočtení cílových kartézských souřadnic a úhlů natočení ve formátu kvartenionů je vypočítána inverzní kinematická funkce MoveIt commanderem. Plán pohybu se dále předává nodu */move_group*.

3.1.4 MoveIt! node move_group

Node */move_group* po inicializaci načte URDF a SRDF modely ramene a následně čeká, až příjme data plánu pohybu. Každý plán je rozdělen na několik mezipozic v celé délce pohybu a ke každé pozici jsou přiděleny úhly natočení jointů

manipulátoru. Ten je následně vykonává v určeném pořadí a tím se docílí plynulého pohybu. Zdroj [6].

3.2 Programovací jazyk a programové vybavení

Pro psaní samotného kódu práce byl zvolen vysokoúrovňový interpretovaný programovací jazyk Python pro jeho jednoduchost, flexibilitu a skvělou čitelnost. Python je objektově orientovaný a na rozdíl od C++ poskytuje dynamickou kontrolu datových typů. Ačkoli je C++ rychlejší, z časového hlediska byl Python adekvátnější volbou.

Jako prostředí pro vývoj softwaru byl zvolen Visual Studio Code od společnosti Microsoft. Je dostupný na operačním systému Ubuntu, což je hlavním kritériem při výběru vzhledem k práci. Poskytuje pomoc při psaní kódu, kontroluje syntaxi, nabízí doplnění rozepsaného řádku, Nabízí širokou škálu možností úpravy jednotlivých částí kódu navíc skrze intuitivní zkratky. V této podkapitole bylo užito zdroje [21].

Pro návrh modelů určených pro 3D tisk byl zvolen CAD software SolidWorks.

3.3 Návrh šachovnice a figurek

Všechny rozměry byly určeny v závislosti na poli působnosti manipulátoru, velikosti end effectoru, jeho úchopové schopnosti, rozlišovací schopnosti kamery a velikosti ochranné klece, ve které je manipulátor umístěn.

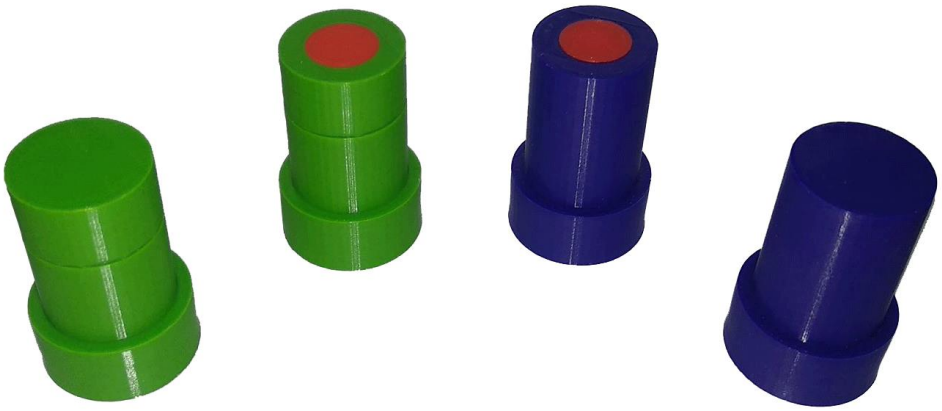
3.3.1 Návrh šachovnice

Šachovnice je vyrobena z dřevěné desky o rozměrech 400x400x20 mm, hrany jsou ošetřeny ABS lepením, ze spodní strany jsou v každém rohu nožičky o výšce 40 mm, celková výška šachovnice je tedy 60 mm. Samotná políčka jsou z papíru o velikosti políčka 42x42 mm.

Šachovnice je indexována od levého horního rohu, kde začíná políčkem X = 1 a Y = 1, kde x představuje vodorovnou osu a y svislou. Pravý dolní roh tedy nese index X = 8 a Y = 8.

3.3.2 Návrh figurek

Podstava figurky je ve tvaru kružnice o průměru 32 mm a výškou 15 mm. Vrchní část je vysoká 35 mm o průměru 26 mm. Tímto je zajištěna vyšší stabilita figurky a zároveň pohodlnější úchop gripperem.



Obrázek 3.2 Herní figurky vyrobené 3D tiskem.

Figurky jsou rozděleny do dvou barev pro dva hráče. První hráč má modré figurky a druhý hráč zelené. Jak je patrné na obrázku Obrázek 3.2Obrázek 3.2 Herní figurky, figurky se ještě dělí na dva typy, tedy jestli figurka představuje pěšáka, nebo dámu. Ty se liší červeným kruhem na vrchu figurky, pokud je přítomen, figurka představuje dámu.

4 REALIZACE

Jak bylo dříve zmíněno, vlastní software je rozdělen do tří programů. Každý má vlastní účel a zpracovává jiná data. Je možno je spustit najednou za pomoci příkazu:

```
$ roslaunch robocheckers robocheckers.launch
```

To má ale nevýhodu, jelikož výstup z konzole je vyhrazen programu spravující uživatelské rozhraní, přičemž zbylé dva programy své výstupy zapisují do souborů. Druhou variantou, která tento problém řeší, je spuštění všech tří programů postupně jednotlivými příkazy:

```
$ rosrun robocheckers game_detection.py
$ rosrun robocheckers game_interfacen.py
$ rosrun robocheckers robot_move.py
```

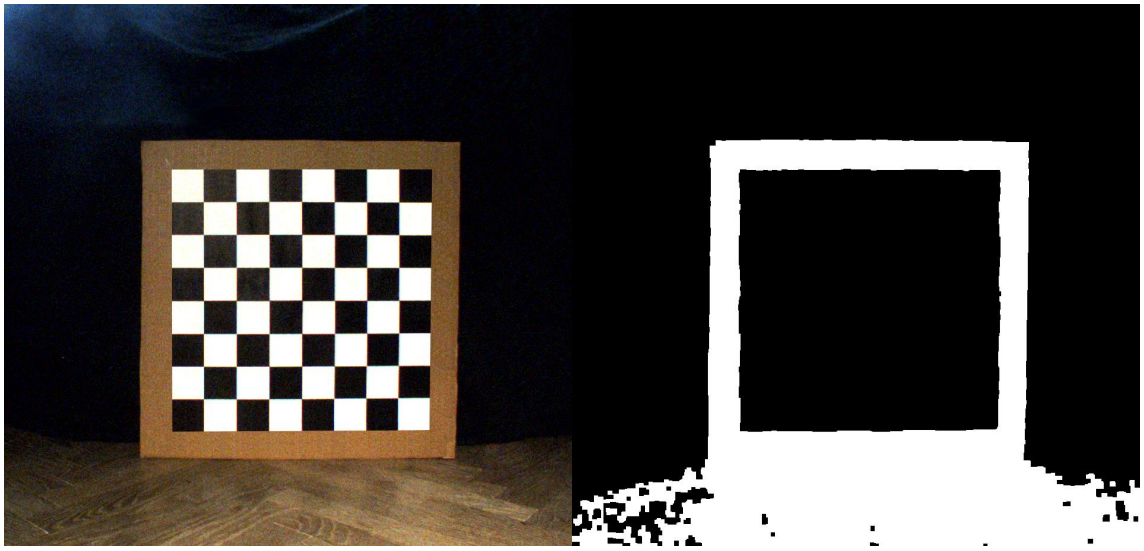
Tímto způsobem je jednodušší debug a samotný vývoj. Před zapnutím vlastního softwaru je zapotřebí spustit buď simulaci, nebo se připojit na reálné rameno, v každém případě je nutné, aby node */move_group* byl již v provozu a aby na parametrickém serveru již byly nahrány URDF a SRDF specifikace modelu manipulátoru.

4.1 Detekce šachovnice a figurek

Nejprve je na obraz aplikován filtr s určitým barveným rozsahem, který odpovídá okolí šachovnice. Ten je definován HSV barevnou maskou v následujících hodnotách:

```
background = ((0, 50, 20), (40, 255, 255))
```

První ze závorek představuje spodní hranici filtru, zatímco druhá z nich vyznačuje tu horní. Jednotlivá čísla v závorkách představují úhel na barevném kruhu (hue), saturaci a hodnotu osvětlení (value). Tímto způsobem je získán obrázek Obrázek 4.1, ve kterém je potřeba najít hrany šachovnice. Detekuje se nejmenší nalezený čtverec s minimální velikostí strany 400 pixelů. Za předpokladu, že je kamera vzdálená 800 mm od šachovnice a známe její rozlišení, je možno tuto hodnotu jednoduše vypočítat.



Obrázek 4.1 Původní snímek a snímek s aplikovanou maskou.

Této detekce je dosaženo za pomoci OpenCV funkce *findContours()* [25]. Její algoritmus je popsán ve zdroji [26].

```
Python: cv2.findContours(
                image,      # zdrojový binární obraz
                mode,       # zvolený mód extrakce
                method [,   # zvolená metoda aproximace kontur
                contours [, # nalezené kontury
                hierarchy [,# výstupní vector informující o topologii obrazu
                offset]]])  # posunutí kontur o určitou hodnotu
```

Návratové hodnoty funkce jsou jednotlivé kontury a výstupní vektor nesoucí informaci o topologii snímku. Argument *mode* nabízí několik metod extrakce kontur:

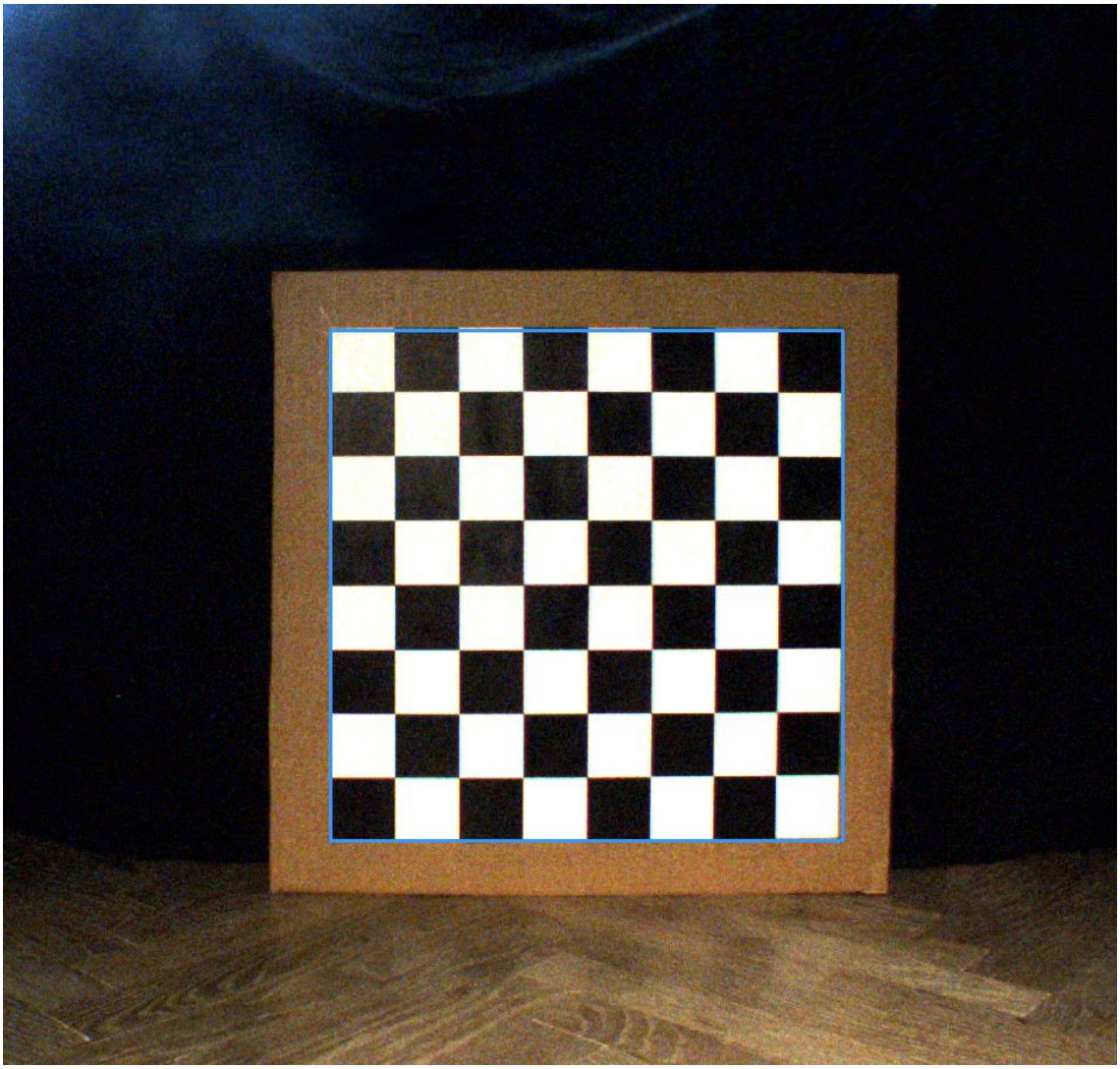
- **CV_RETR_EXTERNAL** – detekce extrémních vnějších kontur.
- **CV_RETR_LIST** – detekce všech kontur bez zásahu do nastavení hierarchie
- **CV_RETR_CCOMP** – detekce všech kontur a jejich následné seřazení do dvou úrovní hierarchie, kde vrchní úroveň obsahuje vnější kontury a spodní úroveň obsahuje vnitřní ohraničení děr externích kontur
- **CV_RETR_TREE** – detekce všech kontur a následné sestrojení hierarchického stromu

Argument *method* umožňuje výběr mezi různými typy aproximace kontur:

- **CV_CHAIN_APPROX_NONE** – funkce vrací všechny nalezené body kontur. Jakékoliv po sobě jdoucí body kontury budou buď horizontální, vertikální, nebo diagonální sousedi.

- **CV_CHAIN_APPROX_SIMPLE** – funkce provádí kompresi na horizontální, vertikální a diagonální segmenty a zanechává pouze jejich koncové body.
- **CV_CHAIN_APPROX_TC89_L1, CV_CHAIN_APPROX_TC89_KCOS** – funkce užívá jeden z Teh-Chin aproximačník algoritmů. [27]

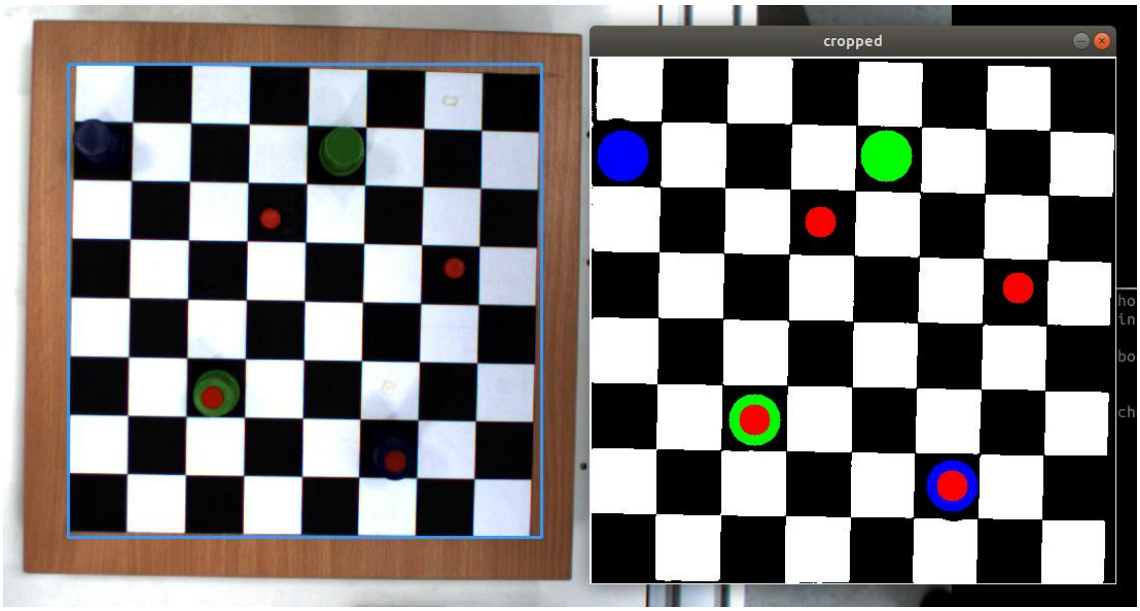
Nyní je známo umístění šachovnice a je možno ji v původním snímku vyznačit funkcí *line()*, která nakreslí přímku mezi dvěma body o zadané barvě. Výsledek lze vidět na obrázku Obrázek 4.2.



Obrázek 4.2 Vyznačená šachovnice po detekci.

Obraz je následně oříznut pouze na daný výběr a upraven na velikost 512x512 pixelů. Na této ploše poté probíhá filtrace jednotlivých barev (modrá – první hráč, zelená – druhý hráč, červená – dáma), čímž se vytvoří 3 nové obrazy nesoucí každý pouze objekty dané barvy. V nich jsou nakonec detekovány figurky

za pomoci OpenCV funkce *HoughCircles()*, která využívá Houghovu transformaci pro detekci kruhů. Podle pozice umístění středu detekovaného kruhu se přiřadí příslušné políčko se souřadnicemi X a Y v rozsahu $< 1, 8 >$. Pokud je detekován na jednom z poliček kruh barvy jednoho z hráčů a k tomu i červený kruh, figurka je považována za dámu. Detekce figurek z originální šachovnice do zrekonstruovaného tvaru lze vidět na obrázku Obrázek 4.3.



Obrázek 4.3 Detekce figurek na šachovnici a jejich následné zakreslení do výřezu.

V tomto okamžiku program přesně ví, kde je šachovnice a kde jsou jednotlivé figurky. Informace o nich se „zabalí“ do ROS Message typu string a posílají se skrze topic */robocheckers/piece* do nodu */game_interface*.

4.2 Herní rozhraní

Jakmile jsou známy pozice figurek, přichází na řadu herní rozhraní, kde se uživateli vypíšou známe pozice figurek, ty jsou samozřejmě také viditelné i na zobrazované šachovnici, kterou neustále obnovuje program pro detekci. Každá figurka představuje instanci třídy *piece*, která má vnitřní proměnné *pos* (list dvou intů udávající pozici) a *rank* (string rozlišující hodnoty „pawn“ a „king“). Tyto instance jsou dále rozděleny mezi dvě instance třídy *player*, které obsahují vnitřní proměnou typu list obsahující právě jednotlivé instance třídy *piece*.

Nyní má uživatel příležitost zadáním souřadnic do konzolové aplikace vybrat figurku pro pohyb. Pokud se na zvolených souřadnicích žádná figurka nenachází, program zahlásí chybu a uživatel je znovu vyzván k zadání validní pozice. Dalším krokem je zadání cílových souřadnic zvolené figurky. Aby nedošlo ke kolizi mezi

dvěma figurkami, je programově ošetřeno, že nelze vybrat destinaci, která již je jinou figurkou obsazena.

V momentě, kdy je cesta naplánována, odešle se zpráva opět typu string přes ROS topic */robocheckers/coordinates* do nodu */robot_move*, zpracují se nová data pozic figurek a znovu se očekává uživatelských instrukcí.

Tato část softwaru slouží jako spojka mezi detekcí herní plochy, uživatelem a programem pro pohyb manipulátoru. Je psán tak, aby nebylo složité na něj navázat a přidat například algoritmus pro hraní s počítačovým oponentem, nebo přidat grafický design pro pohyb figurek.

4.3 Pohyb ramene

Při spuštění se inicializuje instance třídy MoveGroup, kde se načtou data o manipulátoru, nadefinují se a vloží kolizní objekty do okolí ramene a aktivují se dva nody. Prvním z nich je node */robot_move*, který zasílá informace o aktuální pozici manipulátoru pro vizualizaci do RVIZu a také informace o kolizních zdech a objektech do nodu */move_group*. Druhým z nich je *move_group_commander_wrappers*, který na základě funkcí použitých v tomto programu, komunikuje se zbytkem nástavby MoveIt, a tak probíhá plánování a provádění pohybů. Následně je proveden pohyb do strany, aby manipulátor nebránil kameře ve výhledu na šachovnici.

V tomto okamžiku je znám i záměr uživatele, tedy kterou figurkou má v plánu pohnout a na jaké volné políčko. Tyto pozice je potřeba přepočítat na souřadnice v kartézském systému. Jelikož jsou známy pozice rohových polí (Tabulka 4.1), není složité sestavit rovnice pro výpočet.

Tabulka 4.1 Souřadnice rohových polí šachovnice [mm].

X:511 Y:147 Z:243	…	X:511 Y:-147 Z:243
…	…	…
X:243 Y:147 Z:243	…	X:243 Y:-147 Z:243

Z těchto hodnot byly sestaveny rovnice 4.1 a 4.2, kde S_x a S_y jsou souřadnice v kartézském systému v metrech a P_x a P_y představují označení políčka šachovnice.

$$S_x = -0{,}038P_x + 0{,}549 \quad (4.1)$$
$$S_y = -0{,}042P_y + 0{,}189 \quad (4.2)$$

Bohužel tento systém na reálném rameni nefungoval dle očekávání kvůli užití špatného modelu manipulátoru pro výpočet inverzní kinematické úlohy. Celý tento problém je podrobněji popsán v podkapitole 5.1.

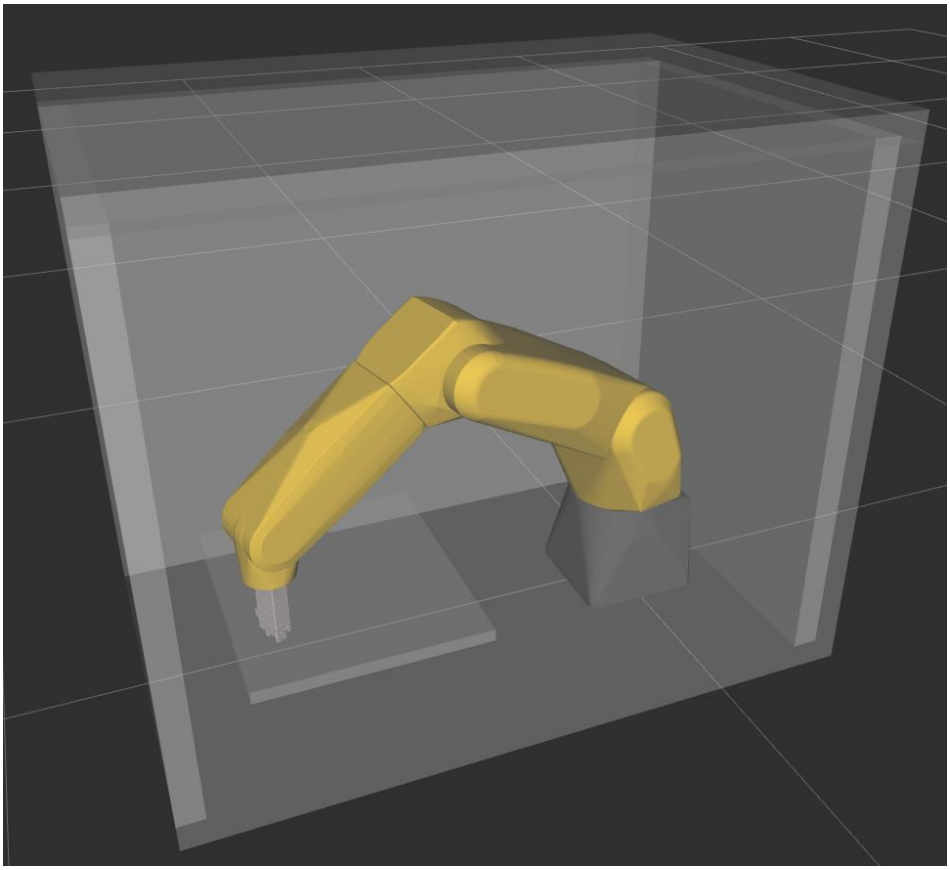
V momentě, kdy jsou přepočítány souřadnice, program za pomoci MoveIt commanderu naplánuje cestu pohybu v jednotlivých sekvencích, pokud je nastavena globální proměnná *DEBUG* na hodnotu *False*, tak se pohyby po naplánování ihned uskuteční, v opačném případě jsou nejprve pohyby simulování v RVIZu a uživatel musí každý naplánovaný pohyb ručně spustit. Sekvence pohybu je následující:

- Pohyb nad políčko se zvolenou figurkou
- Pohyb dolů pro úchop figurky
- Pohyb zpět nahoru nad políčko
- Pohyb nad zvolené cílové políčko
- Pohyb dolů pro položení figurky
- Pohyb zpět nahoru nad políčko
- Pohyb do výchozí pozice na straně

Jakmile se sekvence pohybů dokončí, program vyčkává na nové instrukce. K dispozici je i manuální ovládání, to se nastaví změnou globální proměnné *MANUAL* na hodnotu *True*. Takto program nečerpá hodnoty z ROS Topic kanálu, ale uživateli je umožněno je přímo zadávat do konzole.

4.3.1 Virtuální scéna

Vše, co se odehrává na reálném manipulátoru, je i vykreslováno v simulačním prostředí RVIZ. Pohyb robotického ramene je vykreslován v reálném čase v závislosti na pohybu skutečného manipulátoru. Jestliže je v programu *robot_move.py* nastavena proměnná DEBUG na hodnotu *True*, před každým pohybem se nejprve vykreslí plánovaná trajektorie a následně až se zásahem uživatele je provedena. Tím lze zamezit neočekávanému chování a případnému poškození vlastního okolí při testování.



Obrázek 4.4 Simulace manipulátoru v prostředí RVIZ.

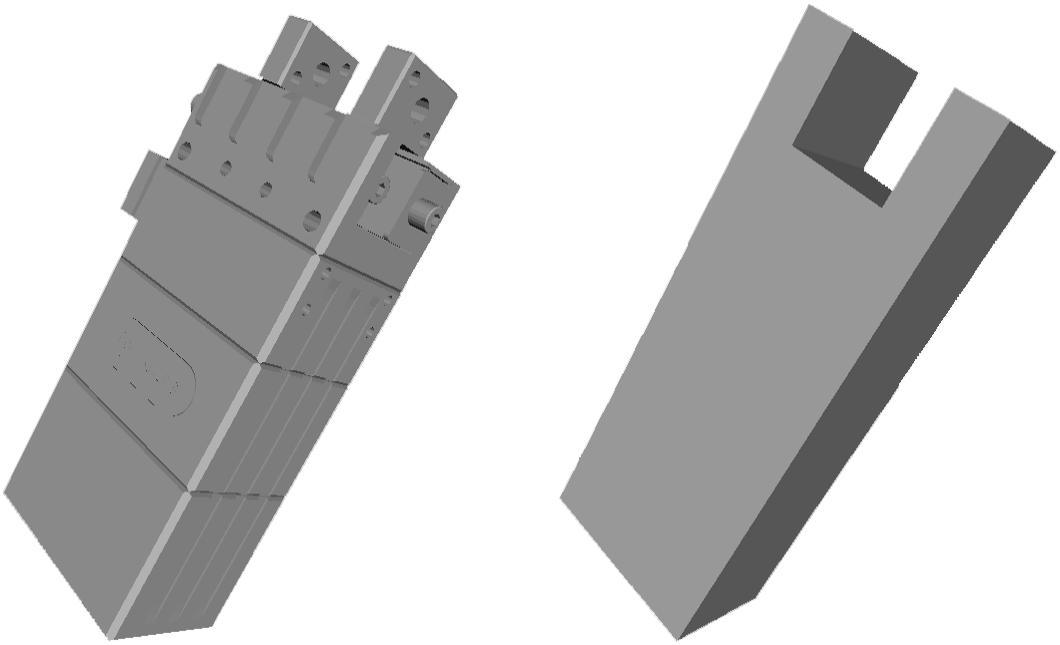
Kolizní zdi

Z obrázku Obrázek 4.4Obrázek 4.4 Simulace manipulátoru v prostředí RVIZ. je patrné, že i v simulaci je manipulátor umístěn do ochranné klece. Ta odpovídá přibližným rozměrům (je nepatrně menší, než reálná) skutečného boxu z plexiskla, kde je rameno umístěno. V simulaci jsou její rozměry nadefinovány následovně: výška = 0,8 m, šířka = 0,68 m a délka = 1 m.

Při plánování trajektorie pohybu manipulátoru jsou tyto zdi zohledněny, tudíž omezují pracovní prostor robotického manipulátoru a zabraňují mu se za ně dostat. Tím je zajištěna bezpečnost pracoviště.

End effector

Robotické rameno je vybaveno gripperem značky Shunk modelu 40-N-N-B, který je potřeba importovat do vizualizace z důvodu kontroly kolizí. Na obrázku Obrázek 4.4 již lze vidět, že je end effector na manipulátoru v simulaci přítomen. Ten byl přidán úpravou URDF a SRDF souborů a přidáním 3D modelů ve formátu STL.

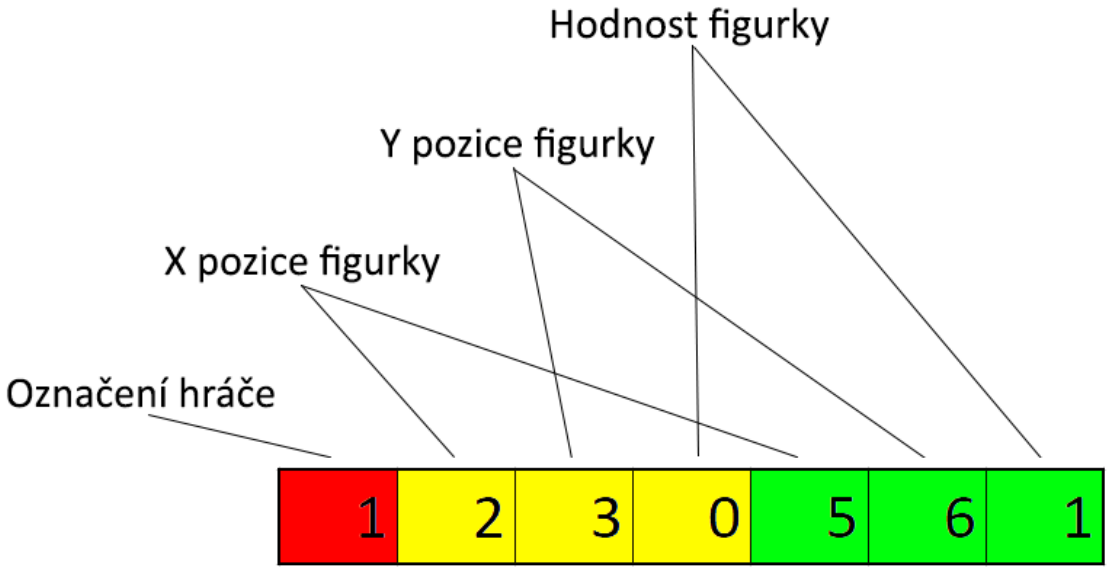


Obrázek 4.5 3D modely gripperů (vlevo vizuální, vpravo kolizní).

Tyto modely musí být dva, přičemž jeden je vizuální a druhý je kolizní. Zatímco vizuální model je pouze k zobrazování v RVIZu, s kolizním modelem se počítá při plánování trajektorie pohybu. Proto je žádoucí, aby byl co nejjednodušší, protože čím je model složitější, tím jsou výpočty časově náročnější. Oba modely jsou k vidění na obrázku Obrázek 4.5. Vizuální model ve formátu STL pochází ze zdroje [28].

4.4 Komunikace v rámci ROSu

Komunikace začíná v nodu */game_detection*, ze které je výstupní zpráva odeslána ve formátu string po ROS topicu */robocheckers/piece*. Zpráva vždy začíná označením hráče, tedy pokud je první znak 1, jedná se o prvního hráče, je-li tento znak 2, jedná se o hráče druhého. Každé další trojčíslí nese informaci o XY pozici figurky a její hodnosti. Na obrázku Obrázek 4.6 lze vidět červeně zbarvený čtverec značící hráče, žluté čtverce pojednávají o první figurce tohoto hráče a zelené o druhé. Každá další nalezená figurka se přidává na konec zprávy.



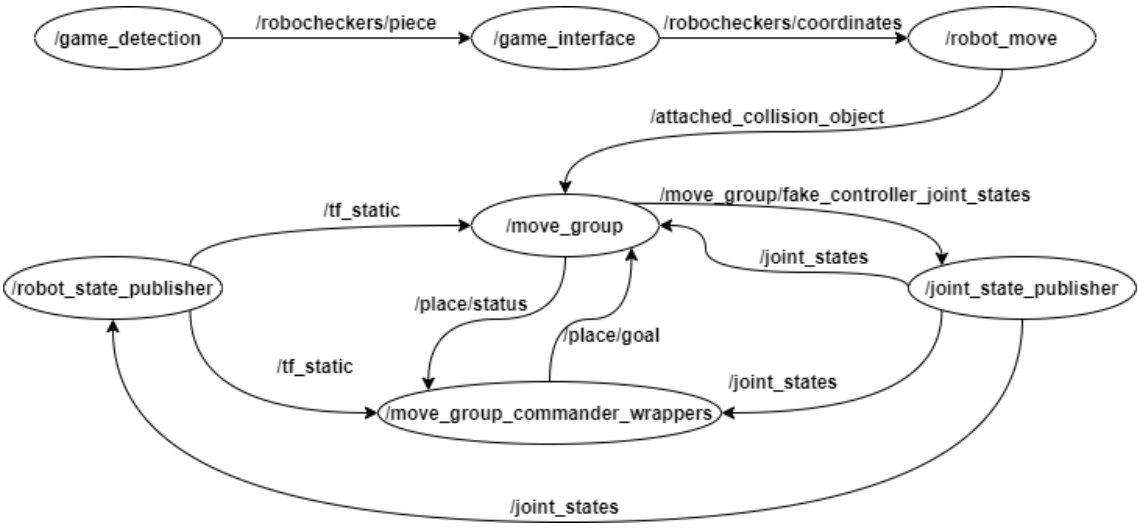
Obrázek 4.6 Vnitřní konstrukce ROS message nesoucí informaci o figurkách.

Po příjmutí těchto dat v nodu */game_interface* a jejich zpracování je vyslána obdobná zpráva po kanále */robocheckers/coordinates*, ve které jsou již pouze čtyři znaky ve formátu string, a to výchozí pozice figurky a cílové souřadnice jejího umístění. Formát zprávy lze vidět na obrázku Obrázek 4.7.

Start		Cíl	
X	Y	X	Y
2	3	4	8

Obrázek 4.7 Vnitřní konstrukce ROS message nesoucí informaci o pozicích.

Tato data putují do nodu */robot_move*, který je převede na kartézské souřadnice. Ranemo je natočeno kolmo dolů a naplánuje se cesta k tomuto cíli za pomoci move_group_commanderu. Tento plán je dále poslán nodu */move_group* v předdefinovaném formátu, kde je zpracován. Následně se vygenerují instrukce pro pohyb ramene a ty jsou distribuovány skrze TCP/IP protokol přímo do kontroléru manipulátoru. Ten nakonec zajistí pohyb ramene. Celá infrastruktura hlavní komunikace lze vidět na obrázku Obrázek 4.8.



Obrázek 4.8 Diagram hlavních nodů.

5 CHYBĚJÍCÍ MODULY

Jelikož byl celý návrh práce obsáhlý, nastaly chyby k jejichž odstranění nedošlo z časových důvodů. Tato kapitola je věnována těm nejvážnějším z nich a nabízí popis možného řešení pro jejich opravu v případě zdokonalování této práce.

5.1 Chybějící vlastní model manipulátoru

Ke správnému výpočtu inverzní kinematické úlohy je zapotřebí mít příslušný model robotického ramene. V této práci je užito fyzického manipulátoru značky Fanuc modelu LR Mate 200iD/4S. Bohužel pro balíčky MoveIt je dostupný nejbližší model LR Mate 200iD, který má mírně odlišné rozměry. Tím pádem při výpočtu inverzní kinematické úlohy dochází k tomu, že výsledná poloha reálného robotického ramene neodpovídá simulaci a celý jeho kartézský souřadnicový systém je nelineární. Odchylka je řádově v jednotkách až desítkách milimetrů.

5.1.1 Návrh řešení

Jelikož model pro reálný manipulátor není k dispozici, nabízí se řešení tvorbou vlastního. Je nejprve zapotřebí všechny linky manipulátoru namodelovat v jednom z dostupných 3D modelovacích CAD systémů a exportovat je ve formátu STL. Pro ulehčení práce je možné namodelovat jen jednu sadu a použít ji jak pro vizualizaci, tak pro počítání kolizí. Dále je potřeba definovat, kde se jednotlivé linky nacházejí a kde jsou spojeny kloubem. Toho lze dosáhnout tvorbou vlastního URDF souboru, kde je mimo jiné potřeba nadefinovat maximální a minimální hodnoty jointů, maximální rychlosti a zrychlení pohybu apod. Dalším krokem by bylo vytvoření SRDF souboru, kde budou popsány možnosti kolizí jednotlivých linků a jointů. Vytvořením a aplikováním tohoto modelu by byl problém zcela vyřešen.

5.2 Chybějící modul ovládající gripper

Implementované balíčeky MoveIt a ROS Industrial neumožňují ovládání I/O konektoru umístěného na robotickém rameni, kde je připojen jeho end effector. Software tedy není schopen pomocí manipulátoru figurku uchytit, toto je velmi závažný problém práce, který se nepovedlo vyřešit taktéž z časových důvodů a kvůli omezenému přístupnosti do laboratoří.

5.2.1 Návrh řešení

Možných řešení je více, některé jsou složitější, jiné náročnější na hardware. Jedním z možných forem realizace může být externí ovládání za pomoci externího hardwaru, jako je například Raspberry Pi, na kterém by byl spuštěn ROS node

propojený skrze TCP/IP protokol na hlavní počítač a čerpal by skrze určity topic informace, kdy má gripper sepnout a kdy rozepnout. Napětí výstupů na Raspberry Pi je 3,3 V a digitální vstupy kontroléru jsou spínány proti potenciálu 24 V, který se nachází na stejném konektoru. Spínání by šlo docílit užitím tranzistorovým spínačem, ale z hlediska ošetření havarijních stavů a bezpečnosti je vhodnější toto ovládání galvanicky oddělit například optočlenem. V hlavní smyčce programu *ROS_MOVESM*, který je spuštěn na kontroléru manipulátoru, je následně potřeba přiřadit tento ovládaný vstup na výstup, kterým je ovládán gripper.

ZÁVĚR

V první kapitole je čtenář seznámen s fungováním platformy ROS, s děním na pozadí komunikace a užitých nástrojů, jako jsou například vizualizační prostředí RVIZ, nástavba ROS Industrial, nebo balíček MoveIt. Dále je popsána nejen teorie ohledně robotických manipulátorů včetně rozboru inverzní kinematické úlohy, ale je zmíněno i použité robotické rameno a jeho kontrolér.

Druhou kapitolou se dostáváme do tajů zpracování obrazu, jednotlivých algoritmů vyhledávající objekty, jako je například kruh nalezený v hranovém obraze pomocí Houghovy gradientní metody. Je probrána i barevná filtrace a úpravy obrazu s ní související. Následně je zmíněno OpenCV, jako knihovna implementující tyto algoritmy pro programování v jazyce python, a nakonec užitá kamera spolu s objektivem.

Ve třetí kapitole se už rýsuje jistý návrh řešení úlohy, který rozebírá celou problematiku na dílčí problémy a kde a jak se tyto problémy budou řešit. Jsou popsány funkce jednotlivých nodů. Tato kapitola plynule přechází v kapitolu čtvrtou, kde přichází podrobný rozbor.

Čtvrtá kapitola odborně rozebírá realizaci zpracování obrazu a jeho užití pro detekci šachovnice, figurek, jejich pozice, hodnosti a barvy. Následně pojednává o fungování herního rozhraní a možné budoucí nástavbě tohoto programu. V neposlední řadě je popsána funkčnost plánování pohybu robotického manipulátoru, vyhýbání se kolizním objektům a komunikace s kontrolérem. Nakonec je podrobně rozepsána komunikace vlastních programů v rámci ROSu.

Pátá a zároveň poslední kapitola je věnována chybějícím modulům a řešením, která nestihla být implementována. Návrhy na jejich řešení jsou podrobně rozepsány, aby v budoucnu bylo jednodušší jejich uskutečnění.

Výsledný produkt tedy detekuje jednotlivé figurky, ty přiřadí hráčům a rozliší, zdali se jedná o dámu, či pěšce. Z bezpečnostních důvodů je manipulátor uzavřen v ochranné kleci, a tudíž pohyb figurek na šachovnici probíhá pouze z uživatelského rozhraní, kde hráč může zadávat pohyby figurek libovolně po šachovnici.

V průběhů realizace jsem narazil na spoustu problému, jejichž řešení bylo obtížné, pracoval jsem v novém prostředí s novými knihovnami a nástroji a naučil jsem se mnoha novým poznatkům. Z časových důvodů jsem nestihl velké množství funkcí, které jsem chtěl implementovat, jako například ovládání gripperu, algoritmus pro samotnou hru, nebo vytvoření správného modelu manipulátoru. Původně jsem zamýšlel i grafické uživatelské rozhraní, kde se hráči vykreslí pozice jednotlivých figurek a může s nimi pohybovat, následně by tlačítkem svůj tah potvrdil a rameno by vykonalo patřičný pohyb, bohužel nakonec vykreslování

zůstalo u pouhého znázornění figurek ve vyříznutém obrazu šachovnice a konzolové ovládání pohybů. Jistě za zmínku stojí i samotná komunikace mezi nody vlastních programů, kde formát zprávy je ve formě stringů, protože ačkoliv toto řešení funguje, je poměrně daleko od ideálu. Na druhou stranu celý program pro detekci šachovnice a figurek předčil má očekávání a představuje jednou z nejsilnějších stránek celé práce, jelikož funguje i za špatných světelných podmínek až na výjimky. Práci jsem od počátku psal tak, aby se na ni dalo lehce navazovat a jednotlivé moduly zdokonalovat, což je také silnou stránkou vnitřní architektury softwaru.

Literatura

[1] Robotika. *ROS* [online]. Adam Ligocki, 2019 [cit. 2019-11-12]. Dostupné z: https://sites.google.com/site/vutrobotika/navody/ros

[2] Introduction. *ROS Wiki* [online]. [online]. Last modified on 8.8. 2018 [cit. 2019-12-17]. Dostupné z: http://wiki.ros.org/ROS/Introduction

[3] Description. *ROS Industrial* [online]. [cit. 2019-12-25]. Dostupné z: https://rosindustrial.org/about/description

[4] Catkin: Conceptual overview. *ROS Wiki* [online]. 2017 [cit. 2019-12-26]. Dostupné z: http://wiki.ros.org/catkin/conceptual_overview

[5] Getting Started. *MoveIt* [online]. [cit. 2019-12-26]. Dostupné z: http://docs.ros.org/melodic/api/moveit_tutorials/html/doc/getting_started/getting_started.html

[6] Concepts. *MoveIt* [online]. [cit.2019-12-27]. [cit.2019-26-12]. Dostupné z: https://moveit.ros.org/documentation/concepts/

[7] MoveIt Setup Assistant. *MoveIt* [online]. [cit.2020-01-03]. Dostupné z: http://docs.ros.org/melodic/api/moveit_tutorials/html/doc/setup_assistant/setup_assistant_tutorial.html

[8] Industrial. *ROS Wiki* [online]. Last modified on 13.5. 2019 [cit.2020-01-02]. Dostupné z: http://wiki.ros.org/Industrial

[9] Fanuc Experimental. *ROS Wiki* [online]. Last modified on 4.3. 2018 [cit.2020-01-02]. Dostupné z: http://wiki.ros.org/fanuc_experimental

[10] Fanuc LR Mate 200iD/4S: *Technický list* [online]. 2019 [cit.2020-02-06]. Dostupné z: https://www.fanuc.eu/~/media/files/pdf/products/robots/robots-datasheets-en/lr-mate/datasheet%20lrmate-200id-4s.pdf?la=cs

[11] Configuration of the ROS-Industrial driver on Fanuc controllers. *ROS Wiki* [online]. Last modified on 9.10. 2019 [cit.2020-01-04]. Dostupné z: http://wiki.ros.org/fanuc/Tutorials/hydro/Configuration

[12] Fanuc_driver_exp. *GitHub* [online]. Last modified on 20.12. 2019 [cit.2020-03-06]. Dostupné z: https://github.com/gavanderhoorn/fanuc_driver_exp

[13] DFK 41BU02 color camera. *ImagingSource* [online]. 2017 [cit.2020-02-06]. Dostupné z: https://s1-dl.theimagingsource.com/api/2.5/packages/documentation/factsheets-single/fsdfk41bu02/f5cc820e-0a07-57ef-8b29-553271f4a48a/fsdfk41bu02.en_US.pdf

[14] Snímací čipy CMOS vs. CCD. *Digimanie* [online]. Milan Šurkala, 2009 [cit.2020-02-06]. Dostupné z: https://www.digimanie.cz/fotomobily-snimaci-cipy-cmos-vs-ccd/2885

[15] H0514-MP2. *Computar* [online]. [cit.2020-03-06]. Dostupné z: https://computar.com/resources/files_v2/158/H0514-MP2.pdf

[16] GONZALEZ Rafael C., WOODS Richard E. Digital Image Processing (4th Edition). Pearson 2017. ISBN 9780133356724.
[17] Morphological Transformations. *OpenCV docs* [online]. 2020 [cit.2020-02-06]. Dostupné z https://docs.opencv.org/trunk/d9/d61/tutorial_py_morphological_ops.html
[18] Gary Bradski and Adrian Kaehler. Learning OpenCV (1st Edition). O'Reilly Media 2008. ISBN 9780596516130.
[19] RGB kódování barev. *Wikiskripta* [online]. [cit.2020-02-06]. Dostupné z https://www.wikiskripta.eu/w/RGB_k%C3%B3dov%C3%A1n%C3%AD_barev
[20] HSV. *Wikipedia* [online]. Last modified on 7.8. 2019 [cit.2020-02-06]. Dostupné z: https://cs.wikipedia.org/wiki/HSV
[21] About. *OpenCV* [online]. [cit.2020-04-06]. Dostupné z: https://opencv.org/about/
[22] Why VS Code. *Visual Studio Code* [online]. [cit.2020-04-06]. Dostupné z: https://code.visualstudio.com/docs/editor/whyvscode
[23] Inverse kinematics. *Kansas State University* [online]. [cit.2020-04-06]. Dostupné z: http://faculty.salina.k-state.edu/tim/robotics_sg/Arm_robots/inverseKin.html
[24] Kinematika robotů. *MRBT - Robotika* [online]. Martin Kozel, 2013 [cit.2020-04-06]. Dostupné z: http://www.uamt.feec.vutbr.cz/~robotika/2013_MRBT/2013_M06_kinematika.pdf
[25] Structural Analysis and Shape Descriptors. *OpenCV* [online]. [cit.2020-05-06]. Dostupné z: https://docs.opencv.org/2.4/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html?highlight=findcontours#id3
[26] *Topological Structural Analysis of Digitized Binary Images by Border Following.* Suzuki, S. and Abe [1985], K., CVGIP 30 1, pp 32-46
[27] *On the Detection of Dominant Points on Digital Curve.* Teh, C.H. and Chin, R.T. [1989], PAMI 11 8, pp 859-872
[28] CAD data EGP 40-N-N-B. *Schunk* [online]. [cit.2020-05-06]. Dostupné z: https://schunk.com/cz_cs/sluzby/ke-stazeni/cadecad-data/vyhledavani-dat-cad/?tx_sccad_cad[select_by_id]=0310940

SEZNAM SYMBOLŮ, POJMŮ A ZKRATEK

ROS	-	Robot Operating System
ROS-I	-	ROS industrial
RVIZ	-	Nástroj umožňující 3D vizualizaci v ROSu
Package	-	Balíček ROS softwaru, může obsahovat ROS node
Workspace	-	Pracovní adresář
API	-	Application Programming Interface
Joint	-	Kloub spojující dvě části manipulátoru
Link	-	Spojení mezi dvěma klouby manipulátoru
End effector	-	Koncový nástavec manipulátoru
Gripper	-	End effector schopný úchopu
px	-	Pixel (obrázkový bod)
FPS	-	Počet snímků za vteřinu
IDE	-	Vývojové prostředí
Debug	-	Odstraňování chyb programu

Seznam příloh

Celý program a jeho struktury jsou také dostupné na GIT repositáři VUT dostupné z: https://student.robotika.ceitec.vutbr.cz/DPBP/2019_bp_lichosyt_fanucros

#	Název souboru/adresáře		Význam
1	./src/robocheckers/bin/game_detection.py	-	Program pro detekci šachovnice a figurek
2	./src/robocheckers/bin/game_interface.py	-	Program pro komunikaci s uživatelem
3	./src/robocheckers/bin/robot_move.py	-	Program pro plánování pohybu manipulátoru
4	./src/robocheckers/launch/game_detection.launch	-	Soubor pro spuštění programu game_detection
5	./src/robocheckers/launch/game_interface.launch	-	Soubor pro spuštění programu game_interface
6	./src/robocheckers/launch/robot_move.launch	-	Soubor pro spuštění programu robot_move
7	./src/robocheckers/launch/robocheckers.launch	-	Soubor pro spuštění všech tří hlavních programů
8	./src/robocheckers/CMakeLists.txt	-	CMakeLists pro sestavování programu
9	./src/robocheckers/package.xml	-	Informace o balíčku robocheckers
10	./src/robocheckers/setup.py	-	Slouží k instalaci balíčku pythonu do pracovního adresáře
11	./src/fanuc	-	Oficiální balíček MoveItu pro komunikaci s Fanuc kontroléry
12	./src/fanuc_experimental	-	Oficiální balíček MoveItu pro rameno Fanuc LR Mate 200iD