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

ROBOTICKÝ MANIPULÁTOR S VIZUÁLNÍ ZPĚTNOU VAZBOU

ROBOTIC MANIPULATOR WITH VISUAL FEEDBACK

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

AUTOR PRÁCE — Bc. Jan Holba

AUTHOR

VEDOUCÍ PRÁCE — Ing. Tomáš Lázna, Ph.D.

SUPERVISOR

BRNO 2025

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní program **Kybernetika, automatizace a měření**

Ústav automatizace a měřicí techniky

Student: Bc. Jan Holba ***ID:*** 228695

Ročník: 2 ***Akademický rok:*** 2024/25

NÁZEV TÉMATU:

Robotický manipulátor s vizuální zpětnou vazbou

POKYNY PRO VYPRACOVÁNÍ:

Cílem této práce je implementace vizuální zpětné vazby na robotické rameno, které tak bude schopno plnit komplexní manipulační úlohy. Požadavkem je integrace řešení do prostředí Robot Operating System (ROS).

1. Seznamte se s předloženým robotickým manipulátorem, možnostmi jeho řízení a vytvořte demonstrační aplikaci schopnou pohybovat robotem definovaným způsobem.
2. Proveďte rešerši metod pro výpočet inverzní kinematické úlohy, zaměřte se na hotová řešení pro ROS 2.
3. Proveďte rešerši algoritmů a frameworků pro řízení robotu pomocí vizuální zpětné vazby.
4. Popište kinematický model robotu, implementujte zvolené řešení inverzní kinematické úlohy a otestujte jeho funkčnost.
5. Po dohodě s vedoucím práce vyberte vhodnou kameru a integrujte ji na robotický manipulátor.
6. Vyberte metodu a prakticky otestujte detekci značek, případně jednoduchých objektů, na základě obrazových dat z kamery na robotu; ověřte závislost na okolních podmínkách.
7. Implementujte parametrizovatelné řešení pro sledování objektu koncovým bodem manipulátoru a demonstrujte jeho funkčnost pomocí videa.

DOPORUČENÁ LITERATURA:

[1] CHAUMETTE, Francois a HUTCHINSON, Seth. Visual servo control. I. Basic approaches. Online. IEEE Robotics & Automation Magazine. 2006, roč. 13, č. 4, s. 82-90. ISSN 1070-9932.

Termín zadání: 10.2.2025 ***Termín odevzdání:*** 21.5.2025

Vedoucí práce: Ing. Tomáš Lázna, Ph.D.

doc. Ing. Petr Fiedler, Ph.D.
předseda rady studijního programu

Abstrakt

Tato práce se zabývá řízením šestiosého manipulátoru s využitím vizuální zpětné vazby, což robotickému rameni dává schopnost provádět úlohy v dynamicky se měnícím prostředí. Kinematická struktura robotického systému je popsána pomocí Denavit-Hartenberg parametrů a v URDF formátu. V prostředí ROS2 jsou implementovány metody pro řešení přímé i inverzní kinematiky, včetně pokročilého solveru s nullspace optimalizací v singularitě. Požadavky na kloubové pohyby zpracovává vícevláknový ovladač s asynchronním přístupem ke sběrnici. Práce se věnuje řízení s vizuální zpětnou vazbou, konkrétně metodám IBVS a PBVS, přičemž na základě provedené analýzy byla zvolena varianta PBVS. Je rovněž kladen důraz na volbu vhodné kamery, její umístění a kalibraci. K minimalizaci latence při komunikaci v rámci ROS2 systému jsou nasazeny pokročilé techniky jako node composition a intra-process communication. Popsány jsou také konstrukční problémy použitého manipulátoru Arctos, společně s jejich řešením.
Byl také implementován plánovač lineární trajektorie a grafická aplikace pro vizualizaci a konfiguraci systému. Vytvořený systém je prezentován formou demonstračního videa a bude nasazen v rámci projektu Brno Mars Rover.

Klíčová slova

Vizuální zpětná vazba, PBVS, Šestiosý manipulátor, Kinematika, RGB-D kamera

Abstract

The thesis deals with the visual servoing of a six-axis manipulator which grants the manipulator the ability to perform tasks in a dynamically changing environment. The kinematic structure of the robotic system is described using Denavit-Hartenberg parameters as well as URDF format. Methods for solving forward and inverse kinematics are implemented within the ROS2 framework, including an advanced solver with nullspace singularity optimization. Joint motion requests are handled by a multithreaded driver with asynchronous access to the communication bus. Various methods of the visual servoing are introduced with the PBVS approach being integrated into the system. The importance of choosing the right camera type and placement as well as its calibration is emphasized. Advanced optimization methods such as node composition and intra-process communication are implemented with the goal of decreasing the latency during communication amongst ROS2 nodes. Solutions to number of mechanical challenges related to the Arctos manipulator are also discussed. Furthermore, the graphical interface for monitoring and control of the system was implemented alongside a path planning algorithm. The developed system will be deployed as part of the Brno Mars Rover project.

Keywords

Visual servoing, PBVS, Six-axis manipulator, Kinematics, RGB-D camera

Bibliografická citace

HOLBA, Jan. *Robotický manipulátor s vizuální zpětnou vazbou*. Online, diplomová práce. Tomáš LÁZNA (vedoucí práce). Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2025. Dostupné z: https://www.vut.cz/studenti/zav-prace/detail/168269. [cit. 2025-04-20].

Prohlášení autora o původnosti díla

Jméno a příjmení studenta:	*Bc. Jan Holba*
VUT ID studenta:	*228695*
Typ práce:	*Diplomová práce*
Akademický rok:	*2024/25*
Téma závěrečné práce:	*Robotický manipulátor s vizuální zpětnou vazbou*

Prohlašuji, že svou závěrečnou práci jsem vypracoval samostatně pod vedením vedoucí/ho 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: 20. dubna 2025 ----------

podpis autora

Poděkování

Děkuji vedoucímu diplomové práce Ing. Tomášovi Láznovi, Ph.D. za účinnou metodickou, pedagogickou a odbornou pomoc a další cenné rady při zpracování mé diplomové práce. Poděkování patří také Ing. Stanislavu Svědirohovi. Jeho praktické rady týkající se konstrukce manipulátoru byly při řešení mnoha technických problémů klíčové.

V Brně dne: 20. dubna 2025 ----------

podpis autora

Obsah

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

Úvod

Tato práce se zabývá tvorbou aplikace pro řízení robotického ramene na základě vizuální zpětné vazby. Tradiční aplikace robotických manipulátorů často spoléhají na předem definované trajektorie a nejsou schopné efektivně pracovat v dynamicky se měnícím prostředí. Vizuální zpětná vazba je proto důležitým aspektem robotické autonomie a napomáhá celkové robustnosti systémů pro manipulační úlohy. Díky vizuální zpětné vazbě robotické rameno nemá problém manipulovat s pohybujícími se objekty. Stejně tak není závislé na precizním umístění cíle, například při kolaboraci s člověkem.

Práce klade důraz na univerzálnost jak softwarového, tak hardwarového řešení. Z toho důvodu je nasazeno šestiosé robotické rameno, které svou strukturou napodobuje lidskou paži a zvládne vykonávat celé spektrum úloh bez nutnosti zásahu do mechanické konstrukce. Adaptace na konkrétní aplikaci často představuje jednoduchou výměnu nástroje. Univerzálnost softwarové části zajišťuje integrace systému ROS2 (Robot Operating System), který představuje modulární framework určený pro vývoj robotických aplikací. Výrazně usnadňuje návrh, testování a nasazení komplexních systémů. Jednotlivé části systému spolu komunikují s využitím specifikovaných rozhraní a jsou jednoduše zaměnitelné.

Při řízení s využitím vizuální zpětné vazby je nutné nejprve spolehlivě **detekovat objekt** v prostoru a přesně **stanovit jeho polohu a orientaci**. Tyto informace jsou využity k **výpočtu řídicích zásahů**, které se snaží minimalizovat odchylku od požadované pozice objektu. Na základě vypočtených povelů jsou s využitím **kinematického modelu** řízeny **pohyby jednotlivých kloubů manipulátoru**. Tato práce se bude věnovat všem ze zmíněných aspektů, od detekce objektu až po generování povelů pro kloubové pohony. Pro dosažení požadované přesnosti bude nezbytné provést také hand-eye kalibraci kamerového systému. Veškeré parametry systému budou konfigurovány přes přehledné grafické rozhraní. Práce si rovněž klade za cíl minimalizovat latenci v řídicí smyčce. Budou proto prozkoumány a implementovány různé optimalizační techniky.

Hlavním cílem práce je vytvořit systém, který bude posléze integrován v rámci projektu Brno Mars Rover, kde bude v průběhu soutěže European Rover Challenge vykonávat řadu manipulačních úloh v dynamickém prostředí.

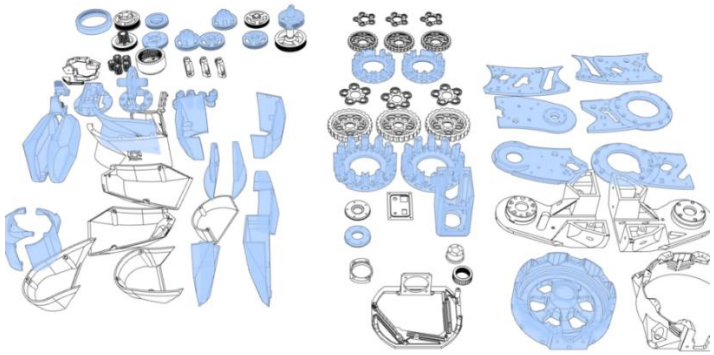
1 MANIPULÁTOR ARCTOS

Pro tuto práci byl využit šestiosý manipulátor Arctos ve verzi V0.2, který je zobrazen v Obr. 1. Tento open source projekt robotického ramene je v současné době stále ve fázi vývoje. Rozvoj projektu podporuje rozsáhlá komunita, která společně aktivně spolupracuje na opravách a vylepšeních.

Obr. 1: Manipulátor Arctos [1]

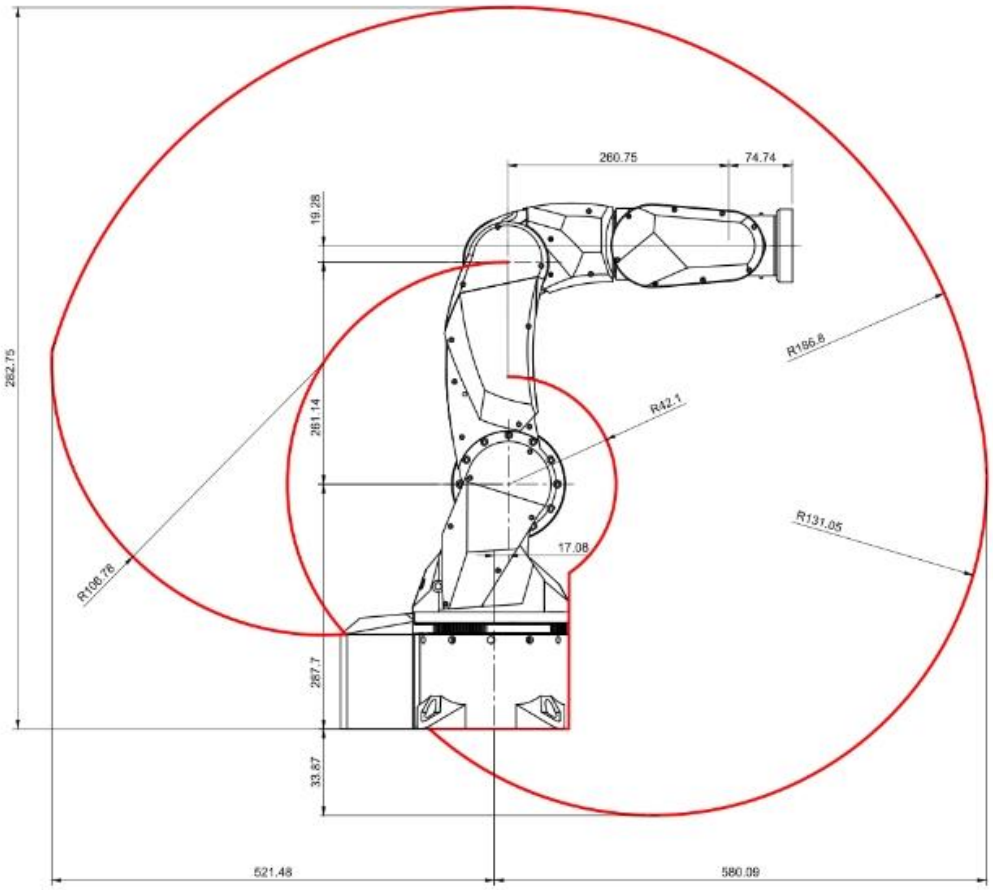
Manipulátor Arctos je označován jako antropomorfní. Jedná se o robotické zařízení navržené tak, aby svým tvarem, strukturou a funkcemi napodobovalo lidskou paži. Tento design umožňuje robotům provádět úkoly podobné těm, které vykonává člověk, a to díky podobné kinematické struktuře a schopnosti manipulovat s objekty v prostředí určeném pro lidskou obsluhu. Adaptace konstrukce na specifickou aplikaci často představuje jednoduchou záměnu nástroje.

Hlavní výhodou, a také důvodem volby tohoto manipulátoru, je skutečnost, že je téměř kompletně vyroben metodou 3D tisku. Veškeré vytištěné díly jsou zobrazeny v Obr. 2. Všechny zbylé díly, které není možné, nebo vhodné, jednoduše vytisknout, jsou běžně dostupné a cenově přijatelné. Tento přístup dává pokročilé technologie do rukou malým výzkumným týmům a laboratořím. To je také jeden z důvodů, proč se kolem projektu vytvořila zmiňovaná rozsáhla komunita. 3D tisk však přináší řadu potíží, kterým se více věnuje kapitola 1.3 Konstrukční problémy manipulátoru.



Obr. 2: Díly manipulátoru vyrobené 3D tiskem [1]

Dokumentace manipulátoru uvádí maximální nosnost 1.3 kg s maximálním dosahem 600 mm. Kompletní rozměry manipulátoru jsou zobrazeny na Obr. 3. Manipulátor je určen pro výukové účely, pro úlohy typu vyzvednutí a umístění a další úlohy automatizace. V komunitě kolem Arctos manipulátoru zatím neexistují žádní oficiální manažeři pro dokumentaci projektu. Poskytnutou dokumentaci píšou kolektivně všichni členové komunity. Obsahuje tak řadu nejasností a nekorektních informací, a proto je potřeba k ní přistupovat obezřetně.



Obr. 3: Rozměry manipulátoru [1]

Výchozím bodem práce byl již sestavený manipulátor s kompletním elektrickým zapojením. Jelikož je ale hardware ramene stále v pracovní verzi, v průběhu vývoje řídicí aplikace se projevila řada poruch, nebo nedokonalostí designu.

1.1 Koubové pohony

Pro pohyb robotického ramene se využívají krokové motory s drivery od čínského výrobce MakerBase. První dva klouby pohání výkonnější pohon s přírubou NEMA23 a driverem MKS SERVO57D. Pro zbylé čtyři klouby je využita méně výkonná varianta motoru s přírubou NEMA17 a driverem MKS SERVO42D. Obě varianty využívají stejné komunikační rozhraní CAN.

Všechny pohony jsou zřetězeny a připojeny k jedinému adaptéru CANable V2.0. Jedná se o malý, cenově dostupný open-source adaptér USB na CAN. CANable se na počítači identifikuje jako virtuální sériový port a funguje jako rozhraní sériové linky na CAN sběrnici. S alternativním firmwarem candleLight se CANable na Linuxu identifikuje jako nativní CAN rozhraní. [2]

Výrobce driverů má své řešení firmwaru bohužel kompletně uzavřené pro veřejnost. Není tak možné zjistit, jaký typ regulátoru pohony využívají a stejně tak nelze regulátor nijak upravit. V průběhu testování to ale nebylo nutné a pohony fungují podle očekávání.

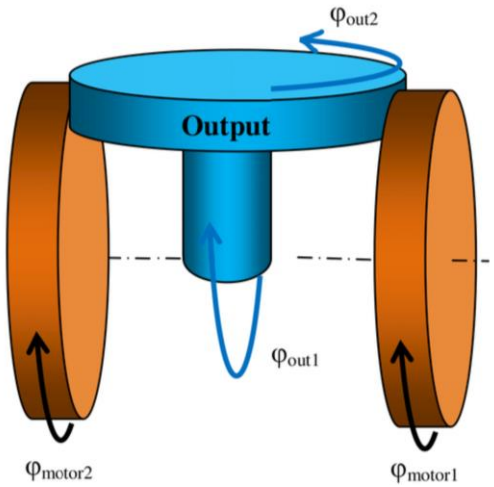
Motory je možné řídit jak v poziční, tak rychlostní smyčce. V poziční smyčce je driveru předáván požadovaný úhel natočení a maximální rychlost daného pohybu. V rychlostní smyčce driver dostává požadavek na rychlost a směr otáčení. [3]

1.2 Diferenciální zápěstí

Manipulátor disponuje diferenciálním zápěstím, které využívá poslední dva motory z řetězce. Tento typ kinematické soustavy využívá diferenciálního mechanismu k transformaci dvojice pohybů na pohyby jiné. Princip mechanismu je zobrazen v Obr. 4. Oranžově jsou zde vyznačeny vstupní členy a modře člen výstupní. Vhodnou kombinací otáčivých pohybů motorů φ_{motor1} a φ_{motor2} získáváme libovolné otáčivé pohyby ve dvou nových osách φ_{out1} a φ_{out2}. Transformace otáčení motorů na otáčení výstupního členu kolem nových os je popsána vztahy (1.1) a (1.2).

$$\varphi_{\mathrm{out1}} = \frac{\varphi_{\mathrm{motor1}} + \varphi_{\mathrm{motor2}}}{2} \tag{1.1}$$

$$\varphi_{\mathrm{out2}} = \frac{\varphi_{\mathrm{motor1}} - \varphi_{\mathrm{motor2}}}{2} \tag{1.2}$$



Obr. 4: Princip diferenciálního mechanismu [4] (upraveno)

Konkrétní využití tohoto mechanismu je zobrazeno v Obr. 5. Červeně jsou zde zvýrazněny dva motory, jejichž pohyby jsou přes zelené převodové ústrojí transformovány do pohybů oranžových kuželových ozubených kol. Modře je poté vyznačeno poslední kuželové ozubené kolo, které představuje výstupní člen mechanismu. Na něj je připevněn nástroj manipulátoru. Tím je v této aplikaci gripper (nástroj pro uchopování) a kamera.

Obr. 5: Diferenciální zápěstí manipulátoru

1.3 Konstrukční problémy manipulátoru

V průběhu vývoje řídicí aplikace se opakovaně objevovaly zásadní konstrukční problémy manipulátoru. Tyto opakované a velmi časté poruchy hardwaru vývoj řídicí aplikace

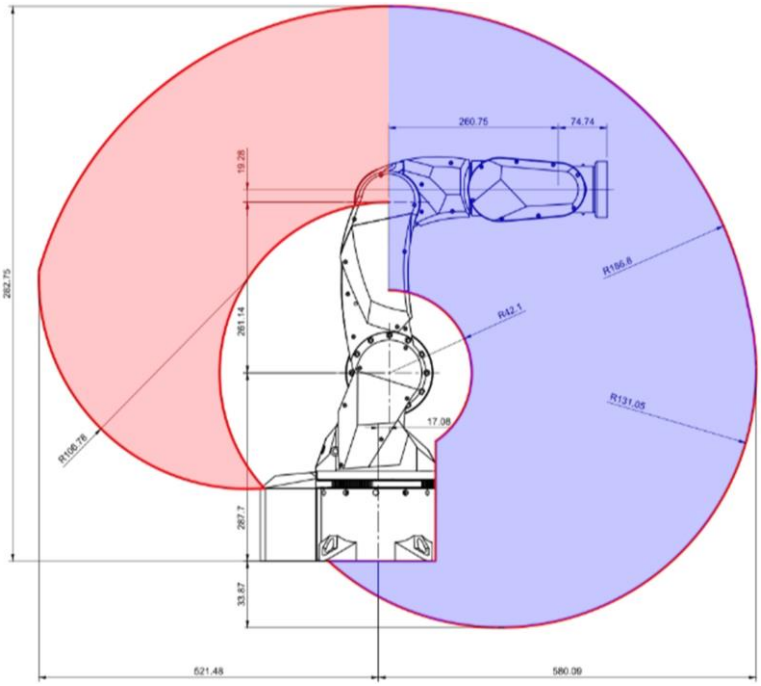
nejen výrazně komplikovaly a zpomalily, ale mnohdy kompletně zastavily. Přestože řešení hardwarových problémů nebylo náplní této práce, představovaly překážku, kterou nebylo možné obejít. **Četné opravy hardwaru si tak ve výsledku vyžádaly značné množství času, původně určeného pro vývoj řídicí aplikace**, a představují podstatnou část této práce.

Všechna omezení manipulátoru pramení ze skutečnosti, že je vyroben metodou 3D tisku. Největší omezení, které 3D tisk přináší, je nízká tuhost a pevnost celé konstrukce ve srovnání s jinými metodami výroby. Díly vyrobené 3D tiskem také nejsou dostatečně precizní a do celého systému vnáší nepřesnosti. Všechny zmíněné nedostatky negativně ovlivňují dynamické vlastnosti manipulátoru. To limituje schopnost manipulátoru reagovat na rychlé změny požadované polohy. Bohužel se jedná o jednu ze zásadních vlastností při řízení s vizuální zpětnou vazbou, které je hlavní náplní této práce. V následujících kapitolách jsou zmíněné slabiny robotického manipulátoru popsány detailně.

K nasazení manipulátoru Arctos v rámci této práce bylo přistoupeno vzhledem k absenci vhodné alternativy, která by splňovala požadavky na otevřenost platformy. Využitý manipulátor umožňuje přímé řízení jednotlivých kloubů a jeho integrace do systému ROS2 probíhá bez větších technických komplikací. Během vývoje této práce na pracovišti nebyla k dispozici žádná komerční platforma, která by splňovala výše uvedené požadavky a manipulátor Arctos byl tak jediným validním kandidátem.

1.3.1 Nízká přesnost

Druhý kloub manipulátoru využívá cykloidní převodovku, která poskytuje vysoký převodový poměr, je velmi robustní a má teoreticky minimální vůli v převodech. Tato nízká vůle je však podmíněna precizní výrobou, kterou 3D tisk bohužel nedisponuje. Alespoň ne v případě, kdy veškeré díly nejsou detailně odladěny pro specifickou tiskárnu. Vůle v převodech, která ve druhém kloubu vznikla představuje v kontextu celého kinematického řetězce výraznou nepřesnost. Pro zajištění alespoň částečně uspokojivé přesnosti je nutné manipulátor provozovat v omezeném pracovním prostoru. Výsledný manipulační prostor ramene se tak sníží přibližně na dvě třetiny. V Obr. 6 je modře zvýrazněn použitelný manipulační prostor ramene.



Obr. 6: Omezený manipulační prostor

Přesnost nastavení pózy koncového bodu manipulátoru omezují také nízké převodové poměry některých kloubů, které jsou zobrazeny v Tab. 1.

Tab. 1: Převodové poměry kloubů

Kloub	Převodový poměr
0	13.6
1	150.0
2	150.0
3	67.8
4	40.0
5	40.0

Nízký převodový poměr omezuje minimální dosažitelnou rychlost kloubu. Klouby tak nejsou schopny realizovat velmi pomalé pohyby, které jsou zásadní pro dosažení precizní pózy manipulátoru při vizuálním řízení.

1.3.2 Nízká nosnost

Jedním z projevů nízké pevnosti manipulátoru je jeho omezená nosnost. Ta je v dokumentaci stanovena na 1,3 kg. [1] Uvedená nosnost nebyla v rámci této práce ověřena z důvodu hrozby poškození manipulátoru a zastavení vývoje. Zároveň v této aplikaci manipulátor nebude uchopovat žádná těžká břemena, takže jej tato vlastnost nijak neomezuje.

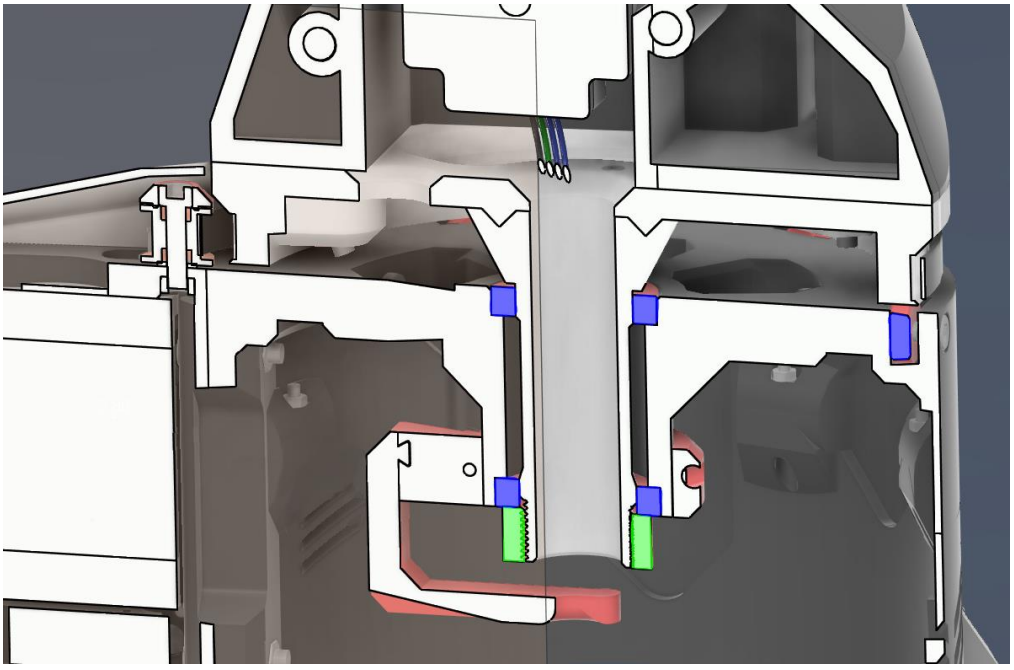
1.3.3 Nízká tuhost

Použité plastové díly disponují nízkou tuhostí, což vede k nežádoucím ohybům konstrukce. Tyto ohyby se projevují v řadě dílů, a to už od základny manipulátoru. V celém řetězci se akumulují a velmi negativně se projevují na výsledné přesnosti nastavení polohy koncového bodu manipulátoru.

Vysoká volnost konstrukce se projevuje nejen v omezené přesnosti nastavení pózy koncového bodu manipulátoru, ale také ve stabilitě veškerých pohybů. Už při rychlostech nižších desítek mm/s se celá konstrukce silně rozkmitá. Přesné a rychlé vizuální řízení je proto velmi omezené. Hrozí také poškození samotné konstrukce.

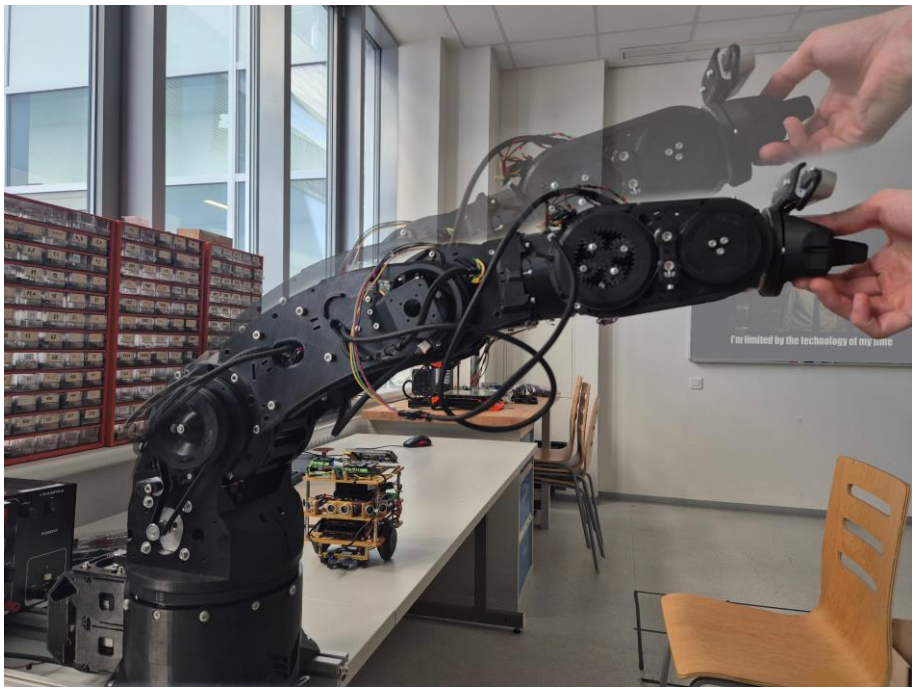
K tomuto nežádoucímu jevu nejvíce přispívá velká vůle v převodech a nevhodné uložení nultého kloubu. Jeho konstrukce je znázorněna v Obr. 7. Konstrukce kloubu je opět kompletně vyrobena z plastu. Celý manipulátor dosedá na sadu menších radiálních ložisek, která jsou rozeseta po obvodu základny a kloub omezují ve vertikálním pohybu. Horizontální pohyb omezují další dvě radiální ložiska s vnitřním průměrem 30 mm. Ložiska jsou v obrázku zvýrazněna modrou barvou.

Problém zde představuje poslední stupeň zajištění kloubu, který je v obrázku zvýrazněn zelenou barvou. Právě tento díl brání manipulátoru, aby přepadl na stranu. S tělem manipulátoru je však spojen přes plastový závit, který není dostatečně pevný na to, aby obstál náporu veškerých sil v systému. Právě tento konstrukční problém vede k výraznému vyklání celé sestavy a představuje jedno z hlavních omezení pro maximální hodnoty ryvu v systému. Náhradou zmiňovaných plastových dílů za kovové by došlo k podstatnému zlepšení dynamických vlastností. Závit by mohl být dotažen podstatně větším momentem a vyklání sestavy by bylo potlačeno.

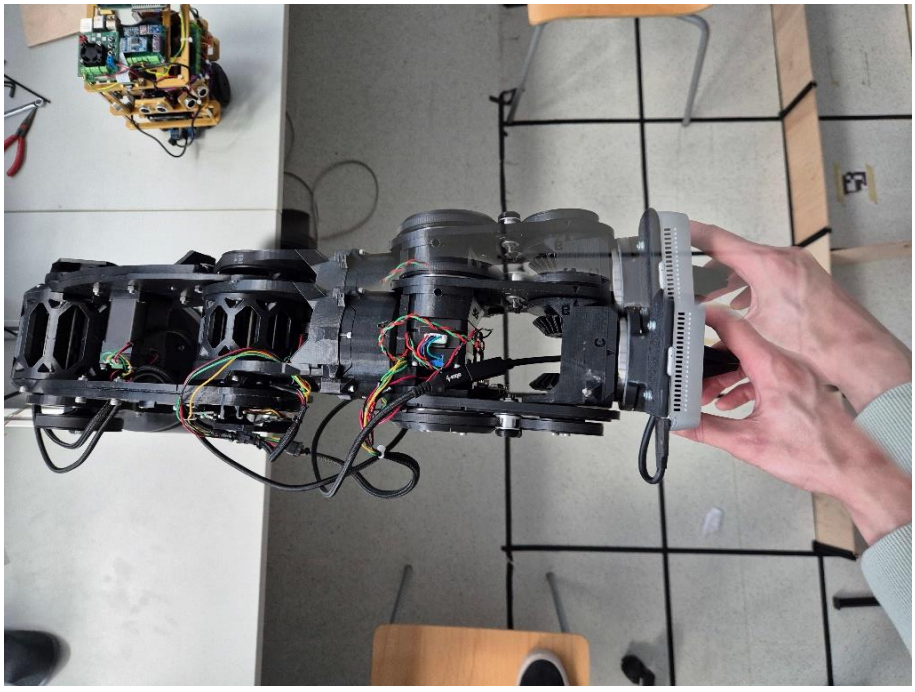


Obr. 7: Konstrukce nultého kloubu

Nízká tuhost celé konstrukce posílená vůlí v převodech kloubů je v rámci celého kinematického řetězce akumulována a ve výsledku přináší podstatnou nepřesnost při polohování koncového bodu manipulátoru. Výchylku způsobenou touto skutečností ilustrují Obr. 8 a Obr. 9. Na manipulátor v okamžiku pořízení fotografií nebyly vyvíjený žádné extrémní síly.



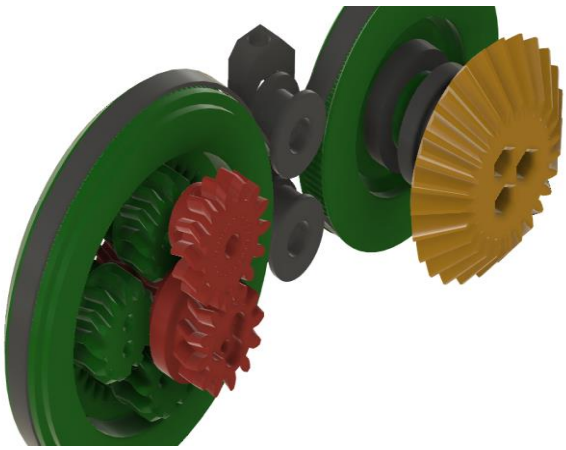
Obr. 8: Volnost konstrukce ve vertikálním směru



Obr. 9: Volnost konstrukce v horizontálním směru

1.3.4 Diferenciální zápěstí

Princip diferenciálního mechanismu je popsán v kapitole 1.2. Hovoří se v ní o tom, že pohyby výstupního členu mechanismu vznikají vhodnou kombinací pohybů vstupních členů. Vstupní členy představují dva krokové motory, jejichž rotační pohyb je transformován přes planetové převodovky a řemenový převod. Sestava jednoho ze dvou převodů mechanismu je zobrazena v Obr. 10.



Obr. 10: Převodové ústrojí pohonu diferenciálního zápěstí

Mechanismus není navržen dostatečně precizně a v průběhu vývoje řídicí aplikace představoval výrazné komplikace. Docházelo k neustálému zadrhávání ústrojí, prokluzu řemene, kolísavému a nerovnoměrném pohybu a častým kompletním blokacím celého mechanismu s následným přetížením motoru. Opravy hardwaru nebyly zadáním této práce, přesto zmíněný problém představoval tak výrazné komplikace, že bylo nutné celé ústrojí několikrát rozebrat a promazat. Tento proces je zobrazen v Obr. 11. Promazání sestavy většinou dočasně problémy mírně potlačilo. Přesto bylo nutné mechanismus znovu vytisknout s nastavením vyšší přesnosti tisku. Poté se situace mírně zlepšila. Sestava ale stále není schopná kompletně plynulého a přesného pohybu.

Obr. 11: Oprava převodového ústrojí diferenciálního zápěstí

Jak již bylo zmíněno, mechanismus manipulátoru jako takový není schopen precizních pohybů. To plyne z metody jeho výroby. U diferenciálního zápěstí je ale tato skutečnost umocněna skutečností, že toto ústrojí stojí na přesné spolupráci dvou nezávislých motorů. Je proto zásadní, aby se oba vstupní členy chovaly naprosto totožně. V opačném případě není možné výstupní člen řídit s požadovanou přesností.

Pokud budeme například požadovat ustálený rotační pohyb kolem výstupní osy φ_{out2}, je nutné zajistit naprosto totožné, ale opačné rychlosti vstupních členů (viz kapitola 1.2). Vlivem zmíněných nedokonalostí to ale v tomto systému není možné a místo izolované ustálené rotace kolem osy φ_{out2} dostaneme nerovnoměrné kolísající pohyby. Navíc se začne projevovat také pohyb kolem osy φ_{out1}, který je v tomto případě parazitní.

Vlivem prokluzu řemene také dochází ke ztrátě informace o aktuálním natočení výstupního členu. Na Obr. 12 je zobrazen stav, který nastane po několika vteřinách pohybu zápěstím a následném požadavku o najetí na nulovou pozici. Je zde vidět, že dochází k odchylce o desítky stupňů, což představuje jeden z hlavních limitujících faktorů celé konstrukce. Pokud mechanismus ztrácí informaci o aktuálním natočení kloubů a není schopen nastavit požadovanou polohu, jeho řízení přestává být spolehlivé.

Obr. 12: Odchylka způsobená ztrátou informace o natočení zápěstí

2 DEMONSTRAČNÍ APLIKACE MANIPULÁTORU

Prvním krokem při implementaci řídicí aplikace bylo seznámení s manipulátorem, způsoby jeho řízení a komunikace v rámci ROS2 systému. Vznikla tedy základní demonstrační aplikace schopná pohybovat robotickým ramenem definovaným způsobem.

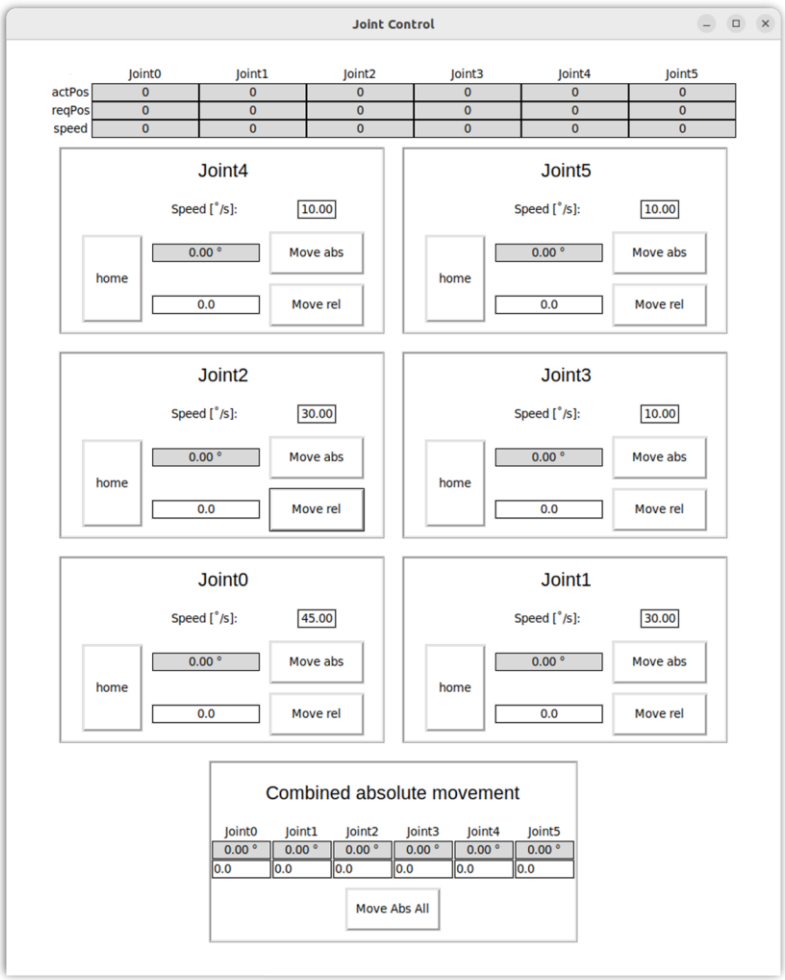
2.1 Výchozí stav softwaru

K projektu bylo poskytnuto pouze samotné robotické rameno a základní ovladač kloubových pohonů, který však vyžadoval podstatné úpravy pro zajištění správné funkčnosti. V prvním kroku bylo nutné identifikovat parametry systému a provést základní konfiguraci všech kloubů. To obnáší nastavení kladného směru otáčení, výpočet převodových poměrů, definování inicializační pozice, limity natočení atd. Původní driver měl také nekorektně definované funkce na čtení stavu. Všechny popsané akce výrazně ulehčilo nasazení základního grafického rozraní, kterému se věnuje kapitola 2.2. V průběhu vývoje řídicí aplikace se ukázalo, že původní ovladač je nevyhovující a bylo nutné ho od základů předělat. Opravy driveru jsou detailně popsány v kapitole 5.2.

2.2 Základní grafické rozhraní

Pro ulehčení procesu testování a ladění pohybů manipulátoru vznikla první verze grafického rozhraní (anglicky Graphical User Interface, dále jen GUI), které umožňuje manuální ovládání robotického ramene na úrovni kloubů. Kompletní GUI ve své první iteraci je zobrazeno v Obr. 13. GUI vzniklo s využitím balíku Tkinter, který je standardní knihovnou v jazyce Python. Díky tomu je to jeden z nejdostupnějších nástrojů svého druhu.

V rámci této práce grafické rozhraní procházelo evolucí, která vedla k postupnému rozšiřování jeho funkcionality. Byly doplněny vyskakovací obrazovky pro řízení kinematiky manipulátoru, stejně tak pro konfiguraci vizuálního řízení a plánování trajektorie. Finálnímu stavu GUI se detailně věnuje kapitola 5.9.



Obr. 13: První verze grafického rozhraní

3 KINEMATIKA

Kinematika se zabývá závislostmi mezi polohami jednotlivých souřadnicových systémů manipulátoru a natočením jednotlivých kloubů. Na robotické rameno Arctos se díváme jako na otevřený lineární kinematický řetězec. Takový řetězec je ukotven v jednom bodě, má jeden koncový bod a mezi těmito body je jediná kinematická cesta. Není rozvětvený a mezi jednotlivými klouby nevznikají smyčky. Opakem je uzavřený nebo rozvětvený kinematický řetězec. [5]

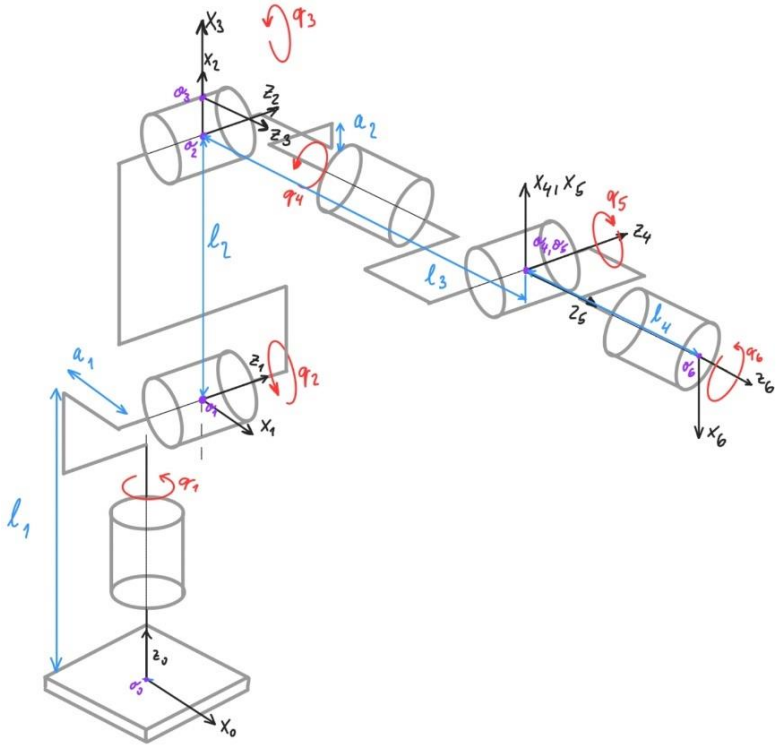
3.1 Formální popis robotické konstrukce

Algoritmy pro výpočet kinematiky potřebují znát konstrukci daného robotu ve správném formátu. V robotice pro popis manipulátoru existuje několik konvencí a definovaných struktur. Formálnímu popisu manipulátoru se věnuje tato kapitola.

3.1.1 Denavit-Hartenbergova konvence

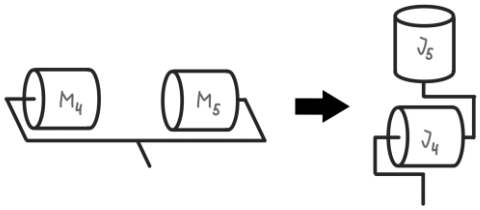
Většina algebraických metod pro výpočet inverzní kinematiky sériového manipulátoru je založena na výčtu Denavit-Hartenbergových parametrů (DH parametry). [6]

DH parametry jsou získány analýzou kinematického řetězce a jsou zapsány do tabulky. Kinematický řetězec se skládá z řady kloubů, které jsou spojeny kostmi. Existuje mnoho druhů kloubů, manipulátor Arctos však využívá pouze klouby rotační. Analýzou struktury manipulátoru byl odvozen kinematický řetězec zobrazen na Obr. 14.



Obr. 14: Struktura manipulátoru

Diferenciální zápěstí, kterému se věnuje kapitola 1.2, v kontextu kinematiky manipulátoru Arctos zajišťuje transformaci motorů M_4 a M_5 na pohyby kloubů J_4 a J_5. Tato transformace je zobrazena v Obr. 15.



Obr. 15: Kinematika diferenciálního zápěstí

Prvním krokem při snaze o popis kinematického řetězce podle DH konvence je stanovení os $\vec{z}$ jednotlivých kloubů. Klouby manipulátoru jsou popsány indexy i. Každá osa $\vec{z}_i$ odpovídá ose rotace daného kloubu i.

Dalším krokem je určení společné normály $\vec{n}_i$ sousedních kloubů. Pokud se osy $\vec{z}_i$ a $\vec{z}_{i-1}$ protínají, je normála $\vec{n}_i$ určena vektorovým součinem daných os. V opačném případě je společnou normálou nejkratší příčka mezi osami.

Dále je nutné určit počátek $\vec{o}_i$ každého souřadnicového systému. Ten je v bodě, ve kterém se protíná $\vec{n}_i$ a $\vec{z}_i$. Nultý souřadnicový systém je připojen k základně robota a je považován za referenční systém.

Osa $\vec{x}_i$ leží na normále $\vec{n}_i$ a její směr si ze dvou možností můžeme zvolit. Nejvýhodnější je volit směr stejný, jako má předešlá osa $\vec{x}_{i-1}$. Díky tomu budou výsledné DH parametry nulové a tabulka se zjednoduší.

Posledním krokem je určení osy $\vec{y}_i$. Ta je definována pravidlem pravé ruky, aby dokončila souřadnicový systém. V diagramu struktury manipulátoru se vektor osy $\vec{y}$ běžně nekreslí, protože jeho směr je jednoznačný a diagram je přehlednější. [7]

Do tabulky DH parametrů jsou zapsány čtyři důležité parametry. Tabulka musí být vyplňována v definovaném pořadí, tj. zprava doleva. Parametr θ_i označuje úhel rotace kolem osy $\vec{z}_{i-1}$ při transformaci ze souřadnicového systému $i-1$ do i. Parametr d_i vyjadřuje translaci (posun) podél osy $\vec{z}_{i-1}$. Totéž platí pro úhel α_i a vzdálenost a_i. Ty jsou ale závislé na ose $\vec{x}_{i-1}$.

Tab. 2: Tabulka DH parametrů manipulátoru Arctos V0.2

Kloub i	**Osa x**		**Osa z**	
	$\boldsymbol{\alpha_i}$ [°]	$\boldsymbol{a_i}$ $[\boldsymbol{mm}]$	$\boldsymbol{d_i}$ $[\boldsymbol{mm}]$	$\boldsymbol{\theta_i}$ [°]
1	-90	$a_1 = 20{,}356$	$l_1 = 285{,}146$	q_1
2	0	$l_2 = 261{,}007$	0	$q_2 - 90$
3	-90	$a_1 = 19{,}911$	0	q_3
4	90	0	$l_3 = 264{,}193$	q_4
5	-90	0	0	q_5
6	0	0	$l_4 = 48{,}826$	q_6

V Tab. 2 vystupují proměnné q_i. Jedná se o aktuální úhel natočení daného kloubu. Tato informace je získána měřením natočení motoru enkodérem a aplikováním převodových poměrů daného kloubu. Jednotlivé úhly q_i je možné zapsat do matice zobecněných kloubových souřadnic $q = (q_1 \ q_2 \ q_3 \ q_4 \ q_5 \ q_6)^T$. Vzdálenosti a a d byly změřeny z 3D CAD modelu, který ke svému manipulátoru Arctos poskytuje [1].

Používání DH parametrů pro výpočet inverzní kinematiky v praxi bývá nepohodlné. Notace DH parametrů není jednoznačná a pro stejnou strukturu robota lze nalézt různé DH parametry, což ztěžuje porovnávání robotů mezi sebou. Orientace souřadnicových systémů základny a koncového efektoru a také počáteční hodnoty kloubových úhlů často nejsou známy, takže vztah mezi strukturou robota a odpovídajícími DH parametry je nutné pracně odvozovat. [6]

Přesto popis pomocí DH parametrů zůstává v robotice oblíbený. Kvůli jeho dlouhé tradici se stal v podstatě standardem. Jiné formáty sice potlačují zmíněné problémy, žádný z nich ale není přijat tak široce jako DH konvence. DH parametry také představují velmi jednoduchý a kompaktní způsob popisu kinematického řetězce. Celý manipulátor je možné popsat pouze sadou čtyř parametrů.

3.1.2 Formát URDF

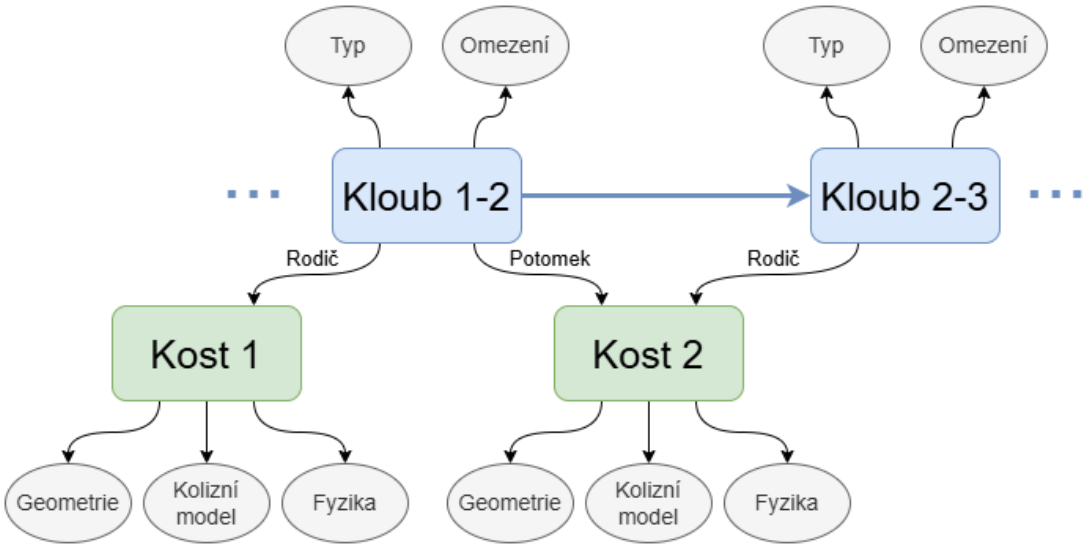
Jeden z alternativních formátů pro popis struktury kinematického řetězce je URDF (Unified Robotics Description Format – Unifikovaný formát pro popis robotů). Jedná se o typ souboru založený na jazyce XML, který vyjadřuje fyzickou strukturu robota. Zahrnuje informace o kloubech, tvaru, velikosti a hmotnosti kostí a dalších fyzikálních vlastnostech robota. K URDF souboru je možné také připojit 3D CAD modely jednotlivých částí robotického systému. Data z URDF souboru poskytují informace o tom, jak robot vypadá a co dokáže, ještě před jeho provozováním. URDF umožňuje přesné modelování a simulaci robotů, přispívá k lepšímu pochopení, plánování a efektivnímu nasazení robotů v různých aplikacích. [8]

V systému ROS2 soubory URDF najdou hned několik uplatnění. URDF vyjadřuje transformace mezi jednotlivými souřadnicovými systémy kinematického řetězce robotu. Toho využívá vizualizační program RViz pro zobrazení aktuální pózy robotu. Stav manipulátoru je možné také simulovat v programu Gazebo.

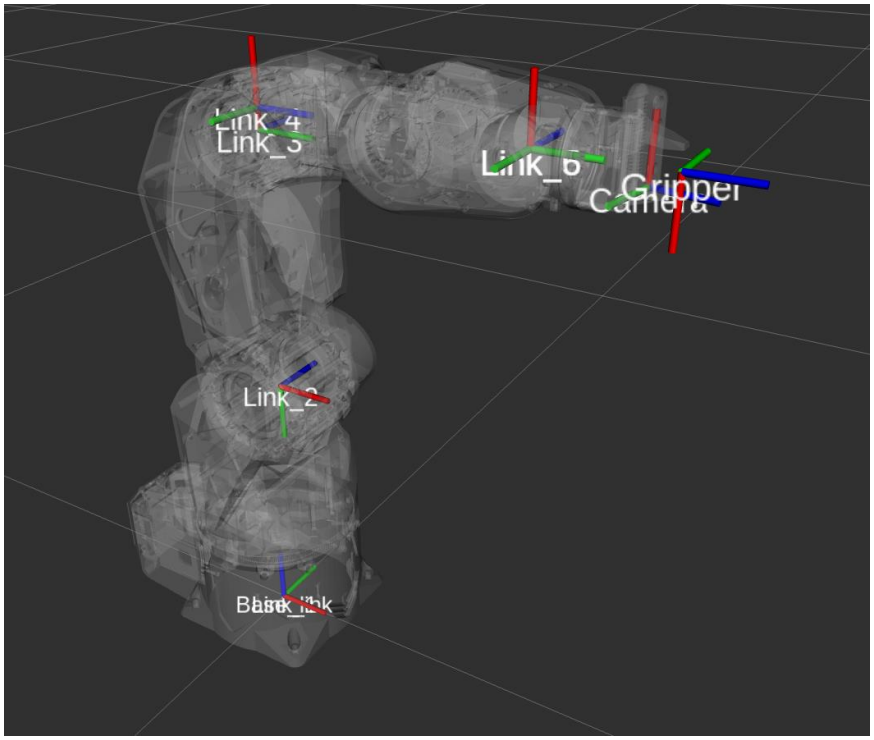
Princip popisu manipulátoru pomocí URDF souboru je zobrazen v Obr. 16. Nejprve je nutné popsat jednotlivé kosti manipulátoru. Je potřeba definovat velikost, tvar a počáteční pózu každé kosti. URDF umožňuje pro vizualizaci kostí definovat jednoduché geometrické tvary. Vzhledem k tomu, že je dostupný 3D model manipulátoru, můžeme využít také ten. Dále definujeme kolizní model a základní fyzikální vlastnosti jako hmotnost a setrvačnost každého dílu. Posledním krokem je definování jednotlivých kloubů, které spojují sousední kosti. Při popisu kloubů je potřeba vyplnit jeho typ a fyzikální a geometrické omezení.

Tento proces se výrazně zjednoduší použitím rozšíření Xacro (XML Macros), které umožňuje v URDF souboru definovat a opakovaně používat šablony pro klouby, kosti, ale také materiály či matematické konstanty. Výsledná kinematická struktura, která vznikla definováním URDF souboru je zobrazena na Obr. 17.

Vytvořený URDF soubor vychází z dokumentace Arctos [1]. Původní soubor měl nekorektně rozložené souřadnicové systémy jednotlivých kloubů a nerespektoval konvenci Denavit-Hartenberga. Celý soubor byl tak podstatně modifikován. Zachována byla pouze kostra původního souboru.



Obr. 16: Struktura URDF souboru



Obr. 17: Výstup URDF souboru

3.2 Přímá kinematika

Než se začneme přímé kinematice věnovat detailněji, je nutné definovat několik pojmů. Póza je souhrnný název pro popis polohy a orientace souřadnicového systému v prostoru. Matematický zápis je zobrazen rovnicí *(3.1)*. Když hovoříme o póze v kontextu manipulátoru, je tím automaticky myšlena póza koncového bodu manipulátoru (označován jako tool center point – TCP), pokud není explicitně řečeno jinak.

$$\boldsymbol{P} = (x \;\; y \;\; z \;\; \alpha \;\; \beta \;\; \gamma)^T \tag{3.1}$$

Kinematická dvojice vyjadřuje spojení dvou sousedních kloubů v kinematickém řetězci. Transformaci mezi dvěma souřadnicovými systémy popisujeme maticí rotace $\boldsymbol{R}$ a vektorem translace $\boldsymbol{T}$. Ty často kombinujeme do tzv. matice homogenní transformace $\boldsymbol{H}$. Tato matice má ve 3D prostoru rozměr 4x4 a její tvar je popsán výrazem *(3.2)*.

$$\boldsymbol{H_{ij}} = \begin{pmatrix} \boldsymbol{R} & \boldsymbol{T} \\ \boldsymbol{p} & m \end{pmatrix} \tag{3.2}$$

Kde:

$\boldsymbol{R}$ – Matice rotace $\quad R = \begin{pmatrix} r_{11} & r_{12} & r_{12} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix}$

$\boldsymbol{T}$ – Vektor translace $\quad T = (x \quad y \quad z)^T$

$\boldsymbol{p}$ – Perspektiva $\quad p = (p_1 \quad p_2 \quad p_3)$

m – Měřítko

Transformace mezi jednotlivými kinematickými dvojicemi můžeme odvodit z tabulky DH parametrů rovnicí *(3.3)*. Tabulka DH parametrů je popsána v kapitole 3.1.1.

$$\boldsymbol{H_{ij}} = \boldsymbol{R_z}(\theta_j) \cdot \boldsymbol{T_z}(d_j) \cdot \boldsymbol{T_x}(a_j) \cdot \boldsymbol{R_x}(\alpha_j) \tag{3.3}$$

Matice R a T vyjadřují definované matice rotace a translace.

$$\boldsymbol{R_z}(\theta) = \begin{pmatrix} \cos(\theta) & -\sin(\theta) & 0 & 0 \\ \sin(\theta) & \cos(\theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad \boldsymbol{T_z}(d) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\boldsymbol{R_x}(\alpha) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) & 0 \\ 0 & \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad \boldsymbol{T_x}(a) = \begin{pmatrix} 1 & 0 & 0 & a \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Pokud je nějaká z proměnných a, d, α, θ variabilní, jedná se o tzv. kloubovou souřadnici, kterou označujeme písmenem q_i. Toto označení zastřešuje úhel natočení rotačního kloubu a posun kloubu lineárního.

Přímá kinematika převádí kloubové souřadnice na souřadnice kartézské. Jinými slovy říká, jakou pózu v prostoru má TCP, jestliže známe kloubové souřadnice. Při řešení přímé kinematiky usilujeme o vyjádření matice $\boldsymbol{H_{BE}}$, která vyjadřuje transformaci mezi TCP

a bází. U otevřeného lineárního kinematického řetězce je matice $\boldsymbol{H_{BE}}$ získána prostým násobením dílčích $\boldsymbol{H_{ij}}$ mezi vedlejšími klouby i a j, jak popisuje rovnice *(3.4)*.

$$\boldsymbol{H_{BE} = H_{B1} \cdot H_{12} \cdot H_{23} \cdot H_{34} \cdot H_{45} \cdot H_{5E}} \qquad (3.4)$$

Výsledek roznásobení všech matic homogenních transformací v této práci není uveden z důvodu přehlednosti. Bez dosazení konkrétních hodnot q_i by zabral několik stran. Stále platí, že matice $\boldsymbol{H_{BE}}$ je matice homogenní transformace, a tudíž má tvar *(3.2)*. Přímá úloha kinematiky tedy hledá takovou funkci $\boldsymbol{f}$, pro kterou platí rovnice *(3.5)*. Takové řešení existuje za každých podmínek a je jen jedno.

$$\boldsymbol{P = f(q)} \qquad (3.5)$$

3.3 Inverzní kinematika

Úloha inverzní kinematiky je obecně složitější úlohou než kinematika přímá. Provádí transformaci z kartézských souřadnic na souřadnice kloubové. V mnoha případech nemá žádné řešení, jindy jich poskytuje nekonečně mnoho. Úkolem inverzní úlohy kinematiky je na základě požadované pózy TCP vypočítat takové úhly natočení jednotlivých klubů, které povedou k dosažení této pózy. Obecně lze problém inverzní kinematiky řešit buď algebraickým, numerickým iterativním nebo geometrickým přístupem.

3.3.1 Algebraické řešení

Algebraický přístup k řešení IK antropomorfních 6DOF manipulátorů popisuje několik výzkumů. Proces definice algebraického výrazu je velmi náročný, následné výpočty kinematiky ale představují jednoduché a rychlé dosazení do výrazu. Tento přístup však trpí tím, že nabízí několik možných řešení a neposkytuje jasnou indikaci, které z nich je vhodné pro konkrétní konfiguraci robotického ramene. Uživatel se často musí spoléhat na svou intuici při výběru správné odpovědi. [9]

3.3.2 Geometrické řešení

Inverzní kinematiku manipulátoru je možné řešit také geometrickým přístupem s využitím základní trigonometrie. Tento přístup je vhodný převážně pro jednodušší manipulátory s nízkým počtem stupňů volnosti (např 2 – 3DOF). V takových systémech je odvození explicitního výrazu jednoduché a rychlé. Pro složitější kinematické řetězce se ale geometrický přístup stává příliš komplikovaným a nepraktickým. Geometrický přístup poskytuje řešení v uzavřeném tvaru (přímé dosazení) a díky tomu bývá ve výsledku výpočetně velmi efektivní. Není ale univerzální a vyžaduje rozsáhlou analýzu konstrukce robotického ramene. [9]

3.3.3 Numerické iterativní řešení

Numerické iterativní metody jsou výpočetně náročnější, protože využívají iterativního přístupu. Navíc nezaručují konvergenci ke správnému řešení. Ze zmíněných možností

jsou ale nejvíce univerzální a jejich nasazení je nejjednodušší.

Většina numerických metod pracuje s Jacobiho maticí $\boldsymbol{J}$. Jedná se o matici parciálních derivací pózy koncového bodu manipulátoru $\boldsymbol{P}$ podle jednotlivých kloubových souřadnic $\boldsymbol{q}$. Tato matice značí, jaká je míra změny pózy TCP při změně natočení jednotlivých kloubů. Kloubové souřadnice jsou zde značeny písmenem $\boldsymbol{q}$, protože klouby mohou být obecně rotační s úhlem natočení θ, nebo lineární s polohou d. Manipulátor Arctos má všechny klouby rotační. Ze vztahu *(3.6)* je zřejmé, že pro výpočet inverzní kinematiky je nutné nejprve využít také výpočtu kinematiky přímé pro určené pózy $\boldsymbol{P}(\boldsymbol{q})$.

$$\boldsymbol{J}(\boldsymbol{q}) = \frac{\partial \boldsymbol{P}(\boldsymbol{q})}{\partial \boldsymbol{q}} \tag{3.6}$$

Algoritmus numerické metody je inicializován počátečním odhadem kloubových souřadnic $\boldsymbol{q}$. S využitím tohoto odhadu poté počítá přímou kinematiku $\boldsymbol{P}$ a určuje odchylku odhadnuté polohy od polohy požadované $\boldsymbol{\Delta P}$. Na základě této odchylky je vytvořen nový odhad kloubových souřadnic. Tento cyklus se opakuje, dokud odchylka odhadované polohy od polohy požadované neklesne pod zadanou hranici. Detaily postupu se však liší podle různých metod.

3.4 Softwarové nástroje pro řešení kinematiky

V rámci této práce není implementován vlastní algoritmus pro výpočet přímé a inverzní kinematiky. Místo toho je využito hotové řešení, které se běžně využívá v rámci ROS2, jak specifikuje zadání. Důvodem je, že tato řešení jsou univerzální a parametrická. Pro popis kinematického řetězce se předává soubor URDF. Při úpravě konstrukce tak nebude nutné výrazně modifikovat kód, ale dojde jen k jednoduché úpravě URDF popisu. Není tedy vhodné investovat čas do vlastního řešení kinematiky. Efektivnější je využít odladěné algoritmy, které jsou dnes už robotickými standardy.

3.4.1 MoveIt

MoveIt je nejrozšířenější framework pro manipulátory, který poskytuje nástroje pro výpočet inverzní kinematiky, řízení, prostorové vnímání, orientaci a navigaci, plánování pohybu, prevence kolizí a mnoho dalších. Je úzce integrován se systémem ROS2 a umožňuje rychlé nasazení a testování manipulátorů v simulovaném prostředí a následné přenesení do reálného hardwaru. MoveIt je široce využíván ve výzkumu, průmyslu i vzdělávání pro vývoj a implementaci robotických aplikací. MoveIt využívají přední výrobci robotů a manipulátorů jako Fanuc, ABB, KUKA, nebo Boston Dynamics. [10] [11]

Přestože MoveIt často představuje de facto standard pro pokročilé řízení manipulátorů, pro tuto práci nebude nasazen. MoveIt je rozsáhlý ekosystém, který integruje obrovské množství pokročilých funkcí, z nichž většina pro tuto aplikaci není vyžadována. Vzhledem k plánované integraci systému do projektu Brno Mars Rover (viz kapitola 7.4) je požadováno jednoduché řešení kinematiky, což MoveIt nesplňuje.

3.4.2 IKFast

IKFast je výkonný nástroj pro řešení inverzní kinematiky. Automaticky analyzuje jakýkoliv složitý kinematický řetězec, ve kterém hledá běžné vzory. Pomocí nich potom generuje C++ kód pro výpočet analytického řešení. IKFast poskytuje extrémně stabilní řešení, která lze na moderních procesorech nalézt během několika mikrosekund. IKFast nepřijímá popis manipulátoru přímo ve formě URDF souboru. Je nutné použít převodník ze standardního formátu URDF na speciální formát OpenRAVE. [12]

Přestože výsledný C++ kód pro výpočet inverzní kinematiky je velmi rychlý, předchozí automatická analýza systému a samotné generování zmíněného kódu je velmi zdlouhavé. V rámci této práce nedopadly snahy o generování kódu úspěšně. IKFast nedokázal analyzovat systém a vygenerovat řešení ani po několika hodinách a jeho běh byl přerušen.

3.4.3 Orocos KDL

KDL (Kinematics and Dynamics Library) je knihovna vyvinutá v rámci projektu Orocos, která poskytuje nezávislé nástroje pro modelování a výpočty kinematických řetězců. Je využívána v robotice pro analýzu a řízení pohybu robotických systémů. KDL nabízí rozsáhlou sadu nástrojů pro práci s geometrickými objekty a umožňuje efektivní transformace mezi různými souřadnicovými systémy. KDL představuje jednu z možných voleb kinematického solveru, který při své konfiguraci nabízí MoveIt. [11] [13] [14]

3.5 Integrace Orocos KDL

Z výše uvedených softwarových nástrojů pro výpočet přímé a inverzní kinematiky byla pro potřeby této práce zvolena knihovna Orocos KDL. Jelikož naše aplikace nevyžaduje všechny pokročilé funkce, které nabízí ekosystém MoveIt, použití samostatného KDL je dostačující a ve výsledku více efektivní.

V KDL nejsou kinematické struktury popisovány tabulkou DH parametrů, ale reprezentují je objekty typu KDL.Chain. Tyto kinematické řetězce není nutné vytvářet ručně, protože existuje ROS balík kdl_parser [15], který kinematický řetězec generuje na základě URDF souboru nahraného při spuštění. V rámci řešení byla doplněna vlastní funkcionalita, která z URDF popisu manipulátoru extrahuje definované limity kloubů. Ty jsou poté využity při řešení úloh kinematiky. Díky tomu je balík kinematiky univerzální a při jakýchkoliv modifikacích manipulátoru je nutné upravit pouze samotný URDF soubor. Vlastní implementaci balíku kinematiky detailně popisuje kapitola 5.3.

4 ŘÍZENÍ S VIZUÁLNÍ ZPĚTNOU VAZBOU

Cílem řízení s vizuální zpětnou vazbou (Visual Servoing) je vybavit robota aktivním vizuálním systémem a zvýšit tak míru jeho autonomie. Takový robot je ve výsledku vysoce robustní vůči změnám v prostředí. Díky tomu je možné jej nasadit do dynamického, nebo předem neznámého prostředí.

Vizuální zpětná vazba je využita v uzavřené řídicí smyčce a napomáhá robotu dosáhnout požadované pózy. Tato technika řízení spoléhá na vizuální data, která jsou online sbírána z pracovního prostoru robota. Povaha těchto vizuálních dat je klíčová pro definici typu vizuálního řízení, jak bude popsáno v následujících kapitolách. [16] [17]

4.1 Metody vizuálního řízení

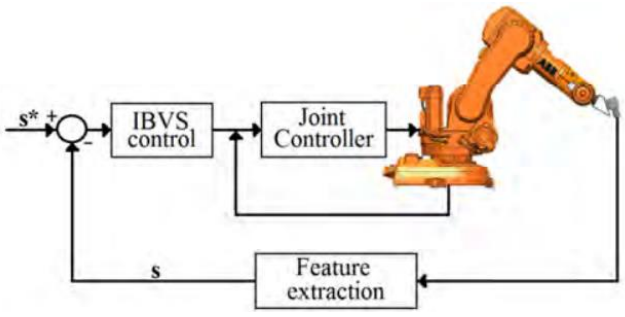
V rámci vizuálního řízení ze sledovaného objektu extrahujeme vizuální charakteristiky $\boldsymbol{s}$ a porovnáváme je s požadovanými vizuálními charakteristikami $\boldsymbol{s}^*$. Všechny metody vizuálního řízení se snaží minimalizovat odchylku $\boldsymbol{e}(t)$. Ta je definována rovnicí (4.1).

$$\boldsymbol{e}(t) = \boldsymbol{s}(\boldsymbol{m}(t), \boldsymbol{a}) - \boldsymbol{s}^* \tag{4.1}$$

Vektor $\boldsymbol{m}(t)$ zahrnuje data získaná měřením v obraze. Vektor $\boldsymbol{a}$ popisuje dodatečné informace o systému (např. intrinsické parametry kamery, 3D model objektu aj.). Jednotlivé metody vizuálního řízení se liší hlavně ve způsobu, jakým jsou tvořeny vizuální charakteristiky $\boldsymbol{s}$ a jaké vizuální informace jsou k tomu využity. Existují dvě základní metody. Při využití metody IBVS jsou charakteristiky $\boldsymbol{s}$ tvořeny sadou významných bodů z 2D obrazu. Metoda PBVS definuje $\boldsymbol{s}$ jako sadu 3D parametrů, které jsou z obrazu odvozeny. [16] [17] [18]

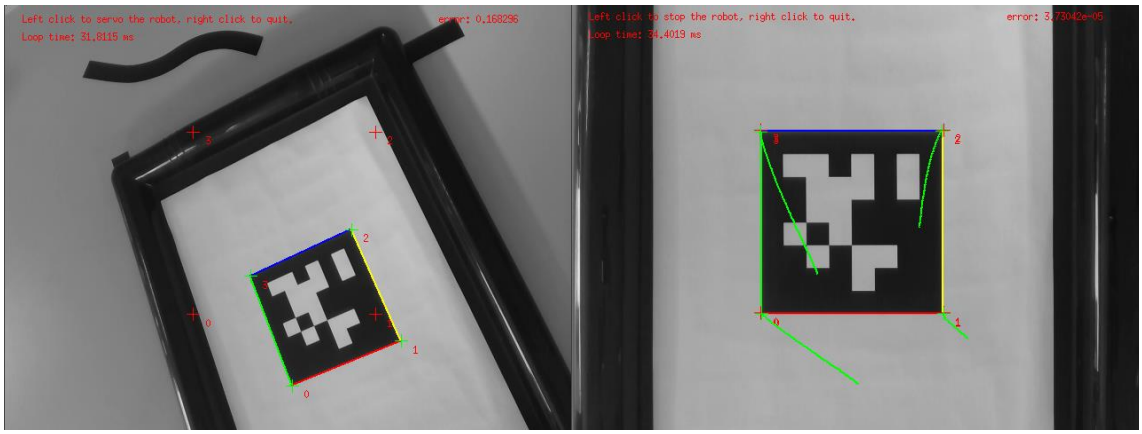
4.1.1 Image-Based Visual Servoing – IBVS

IBVS je metoda vizuálního řízení založená na datech z dvojrozměrných snímků. Proto bývá označována jako 2D vizuální řízení. Metoda v obraze z kamery vyhledává významné body. Ty stejné významné body jsou nalezeny také ve vzorovém obraze a je spočítána jejich odchylka. Následně je řízena poloha ramene pro minimalizaci vypočtené odchylky. [16] [17] [19] Řídicí smyčku při využití metody IBVS popisuje Obr. 18.



Obr. 18: Řídicí smyčka metody IBVS [16]

Princip funkce je zobrazen v Obr. 19. V obraze z kamery jsou detekovány čtyři významné body. V tomto případě se jedná o rohy AR značky. Jako vzor je předána požadovaná poloha těchto bodů. Cílem je minimalizovat odchylku mezi detekovanými a vzorovými body. V pravém obrázku je zobrazen cílový stav, ve kterém se jednotlivé body překrývají. Taktéž je zobrazena trajektorie oněch bodů.



Obr. 19: IBVS z pohledu kamery. Počátek vlevo, cíl vpravo. [20]

Přesnost metody IBVS není závislá na kalibraci kamera-rameno. Vyplývá to z faktu, že pokud je nulová obrazová odchylka, musí být nulová i kinematická chyba. Nízká citlivost na chyby kalibrace je jedna z hlavních výhod oproti metodě PBVS, která je popsána v další kapitole. Další výhodou metody IBVS je nízká výpočetní náročnost, která vede k vyšší rychlosti reakcí na vizuální podněty. [19]

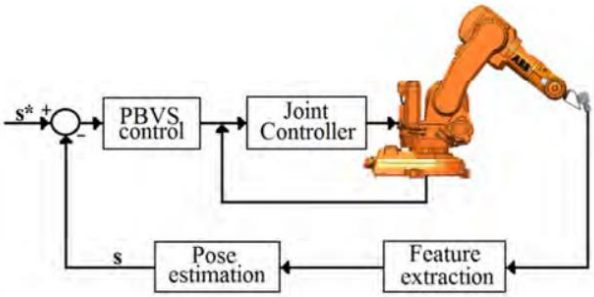
Hlavní nevýhodou IBVS je závislost na viditelnosti významných bodů v obraze. Pokud jsou významné body zakryty nebo mimo zorné pole kamery, například při otočení objektu „zády“ ke kameře, IBVS nemůže korektně fungovat. To vyplývá z faktu, že IBVS je zaměřeno na 2D obrazové parametry a nezajišťuje rekonstrukci geometrie objektu. Úloha je tedy konvergentní pouze v malé oblasti kolem požadované cílové polohy. To znamená, že objekt musí být už na počátku operace v blízkosti požadované pózy. To dělá tuto metodu méně univerzální ve srovnání s PBVS. [19]

4.1.2 Position-Based Visual Servoing – PBVS

Metoda PBVS je založená na odhadu aktuální pózy (polohy a orientace) snímaného objektu ve 3D prostoru. Proto bývá označována jako 3D vizuální řízení. V rámci vizuálního řízení je zadána požadovaná finální póza daného objektu vzhledem ke kameře. Odchylka mezi aktuální pózou a požadovanou pózou objektu je označována jako kinematická chyba, která je definována v kartézském prostoru. Cílem je tuto kinematickou chybu redukovat na nulu. Proces zahrnuje nejprve určení relativního pohybu kamery, který by splnil úkol, a následné provedení daného pohybu manipulátorem. [19] Obr. 20 zobrazuje řídicí smyčku při využití metody PBVS.

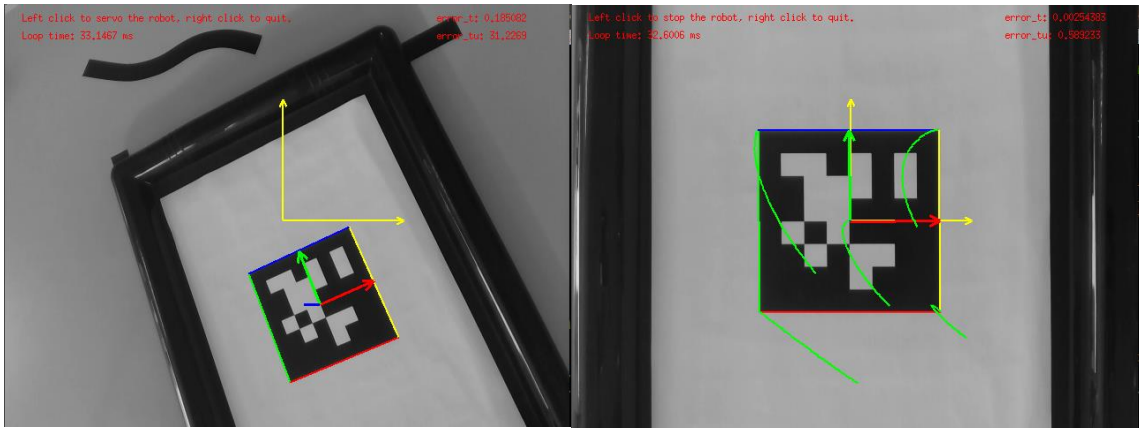
PBVS úzce souvisí s problémem rekonstrukce geometrie scény a odhadu pózy objekty z jednoho nebo více obrazů získaných kamerou. Možné způsoby zahrnují

získávání struktury z pohybu, stereo rekonstrukce aj. Další variantou je také využít hloubkovou kameru. [19]



Obr. 20: Řídicí smyčka metody PBVS [16]

Princip PBVS je zobrazen v Obr. 21. Na rozdíl od metody IBVS zde nejsou detekovány jednotlivé body, ale objekt jako celek. Na základě tvaru objektu v obraze je estimována jeho póza a je mu přiřazen souřadnicový systém. Ten je vyznačen RGB šipkami. Dále jsou v obrázku zobrazeny žluté šipky, které značí osy x a y požadované konečné pózy objektu. V pravém obrázku je zobrazen finální stav, kdy se souřadnicový systém objektu překrývá s požadovanou pózou.



Obr. 21: PBVS z pohledu kamery. Počátek vlevo, cíl vpravo. [21]

Hlavní výhodou vizuálního řízení založeného na poloze je možnost popsat úkoly v termínech kartézské polohy, což je v robotice běžné. Metody PBVS se zdají být nejuniverzálnějším přístupem k problému, protože podporují libovolnou polohu objektu. Naopak metody IBVS jsou závislé na viditelnosti významných bodů v obraze.

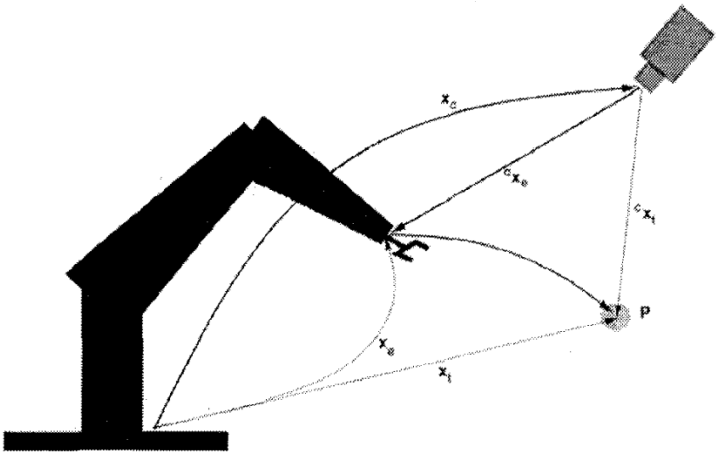
Hlavní nevýhodu PBVS představuje vysoká citlivost na chyby v kalibraci. Zejména u stereo systémů mohou i malé chyby v kalibraci kamer výrazně ovlivnit přesnost polohování. Proto je nutné provézt tzv. Hand-Eye kalibraci, která je popsána v kapitole 4.4.3. Další často zmiňovaná nevýhoda je výpočetní náročnost odhadu pózy objektu. Tento problém je však díky rychlému pokroku v technologii mikroprocesorů už méně významným. [16] [19] [22]

4.2 Umístění kamery

V úlohách vizuální zpětné vazby se běžně používají dvě základní umístění kamery.

4.2.1 Globální eye-to-hand kamera

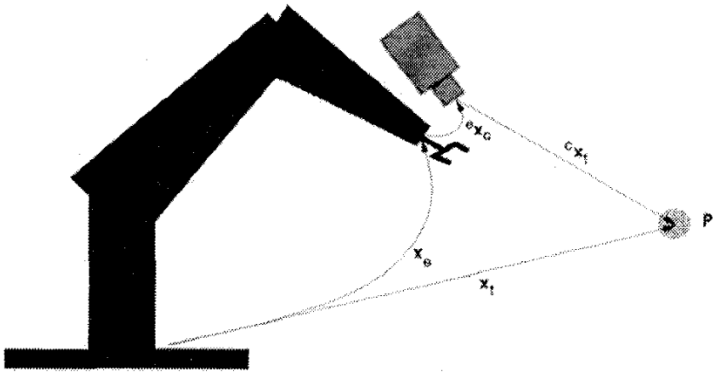
Kamera je umístěna na pevném bodě v pracovním prostoru, jak ukazuje Obr. 22. Její relativní poloha vzhledem k základně manipulátoru je určena jednou a v průběhu operace se nemění. Globální kamera poskytuje kompletní přehled nad celým pracovním prostorem. Postrádá však přesnost, co se týče detailů sledovaného objektu. Taktéž není možné upravovat její zorné pole. [23] [24] [19]



Obr. 22: Eye-to-hand konfigurace [19]

4.2.2 Lokální eye-in-hand kamera

V této konfiguraci je kamera připevněna ke koncovému bodu manipulátoru, jak je patrné z Obr. 23. To vede k omezení jejího užitečného zorného pole. Nedokáže současně sledovat celý pracovní prostor. Připevnění kamery k robotickému rameni jí ale zajišťuje manévrovatelnost a schopnost prohledávat scénu z různých úhlů. Tato konfigurace přináší mnohem větší přesnost v porovnání s globální kamerou, která plyne z toho, že je kamera blíže k cílovému objektu. [19] [23] [24]



Obr. 23: Eye-in-hand konfigurace [19]

4.2.3 Hybridní

Konfigurace eye-to-hand a eye-in-hand mají své specifické vlastnosti. Díky tomu se mohou vzájemně doplňovat. Globální kamera upevněná mimo manipulátor umožňuje detekci předmětu v celém pracovním prostoru robotu. Tato informace slouží pro určení polohy sledovaného objektu a nastavení přibližné polohy robotického ramene. V tom okamžiku řízení přebírá lokální kamera, která poskytuje detailní pohled na objekt. Ta zajistí precizní nastavení polohy manipulátoru. [24]

4.3 Typ kamerového systému

Volba vhodného typu kamery ovlivňuje povahu dat předávaných vizuálnímu řízení. Vlastnosti jednotlivých typů kamerových systémů jsou popsány v následujících kapitolách.

4.3.1 RGB kamera

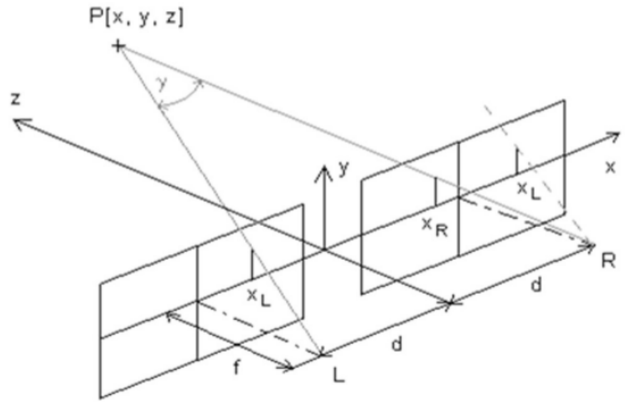
Klasické RGB kamery snímají barevné 2D obrazy ve viditelném spektru světla. Tento typ kamery je vhodný pro IBVS metodu vizuálního řízení, která řídí robota přímo na základě vizuálních dat z dvourozměrného obrazu kamery. RGB však kamera neposkytuje přímé informace o 3D tvaru a vzdálenosti sledovaného objektu, proto se nehodí pro PBVS metodu, která odhaduje pózu objektu v trojrozměrném prostoru. Geometrii objektu je možné rekonstruovat z několika snímků při pohybu kamery. Tento proces je však příliš pracný a pro tuto aplikaci nebude využit.

Pózu objektu je možné odhadnout při využití AR značek o předem definovaných rozměrech. Díky definovaném tvaru značky je potom možné i z 2D snímku odhadnout vzdálenost a orientaci objektu v prostoru. Tento přístup však výrazně omezí univerzálnost celého robota. Vyžaduje totiž umístění značek na předem zvolené objekty. Objekty bez těchto značek není možné sledovat.

4.3.2 Stereoskopické měření

Projekcí trojrozměrného objektu do 2D obrazu dochází ke ztrátě jednoho rozměru – hloubky, resp. vzdálenosti od kamery. Pro vizuální řízení metodou PBVS je nutné tuto informaci obnovit, aby bylo možné rekonstruovat geometrii objektu a estimovat jeho pózu v prostoru. Stereovize provádí rekonstrukci za pomocí dvou samostatných kamer. Tento princip je inspirován lidským zrakem.

Pro správné měření je nutné detekovat stejné významné body ve snímcích z obou kamer, jak je patrné z Obr. 24. Ze znalosti intrinsických a extrinsických parametrů kamer je poté možné dopočítat pozici nalezených bodů v prostoru. Intrinsické parametry zahrnují informace o ohniskové vzdálenosti, velikosti čipu kamery aj. Extrinsické parametry potom popisují vzájemné umístění kamer, jejich vzájemné natočení a vzdálenost základny. [25]



Obr. 24: Princip stereovize [25]

Stereovizní měření je výrazně závislé na precizní kalibraci intrinsických i extrinsických parametrů. Při minimálních odchylkách vznikají velké chyby při rekonstrukci geometrie, z toho vycházející estimace pózy objektu a následném řízení manipulátoru. [19]

Při naší aplikaci je využita lokální eye-in-hand kamera. Kamera je tady umístěna na koncovém bodu robotického ramene. Působí na ni mechanické ryvy a vibrace. To může způsobit rozladění kalibrovaného kamerového systému. Jedná se tak o zásadní nevýhodu této metody měření.

4.3.3 Hloubková kamera

Hloubková kamera poskytuje kromě RGB obrazu také informaci o hloubce, tedy vzdálenosti jednotlivých bodů od kamery. Proto bývá označována jako RGB-D. Na trhu existuje několik různých modelů. Mezi nejčastější se řadí Azure Kinect a Intel RealSense. Kinect využívá princip time of flight. Měří čas mezi vysláním a zpětným odrazem signálu od objektu. Kamery RealSense využívají kombinují princip stereovize a aktivního osvícení IR světelným vzorem. Problémy plynoucí z nutnosti precizní kalibrace stereoskopické kamerové konstrukce jsou zde potlačeny skutečností, že obě kamery jsou uzavřeny v jediném těle. Nutnost přesné kalibrace tak přebírá výrobce. Kombinace stereovize a aktivního osvětlení umožňuje konzistentní výkon i v prostředích s nízkým kontrastem.

Obr. 25: Intel RealSense D435 [26]

4.4 Integrace vizuálního řízení

Robotické rameno použité v rámci této práce je navrženo pro komplexní manipulaci s objekty v trojrozměrném prostoru. Pro schopnost plnění zadaných úloh je nezbytné, aby vizuální řízení bylo dostatečně univerzální. Hlavní výhodou přístupu IBVS je vysoká rychlost reakcí na vizuální podněty. Tato vlastnost zde však není klíčová zejména s ohledem na mechanická omezení manipulátoru. Pro účely této práce byla zvolena metoda PBVS, která zajišťuje řízení na základě požadované pózy objektu ve 3D prostoru, na rozdíl od metody IBVS, která zpracovává pouze 2D obrazová data. Tento přístup tak představuje výrazně flexibilnější řešení.

Pro potřeby této práce byla nasazena kamera Intel RealSense D435, kterou je možné vidět na Obr. 25. Při porovnání konkurenčních modelů tato kamera dosahuje dobrého výkonu při rekonstrukci geometricky složitých objektů, přestože může vykazovat určité nepřesnosti v jemných strukturách. V této oblasti však překonává RealSense D415. Kamera D435 je nejpřesnější v oblasti zkreslení, což ji činí ideální pro aplikace, kde je důležité minimalizovat geometrická zkreslení. [27]

Výpočet geometrie objektu z dvou oddělených kamer vyžaduje stereoskopické algoritmy pro detekci a srovnání bodů mezi dvěma obrazy. Tato operace je výpočetně náročnější než použití RealSense kamery, která tento proces provádí hardwarově. Proto je v této práci upřednostněna RGB-D kamera před dvěma samostatnými kamerami a stereoskopickým měřením.

Volba hloubkové kamery vychází z původního plánu rozpoznávat objekty, porovnávat je s databází CAD modelů a estimovat jejich polohu a orientaci. Kvůli problémům s konstrukcí manipulátoru, kterým se věnuje kapitola 1.3, na takový algoritmus nezbyl čas. Kamera tedy nyní rozpoznává objekty označené AR značkou. Zadání práce však bylo splněno, neboť rozpoznávání objektů byl pouze osobní cíl.

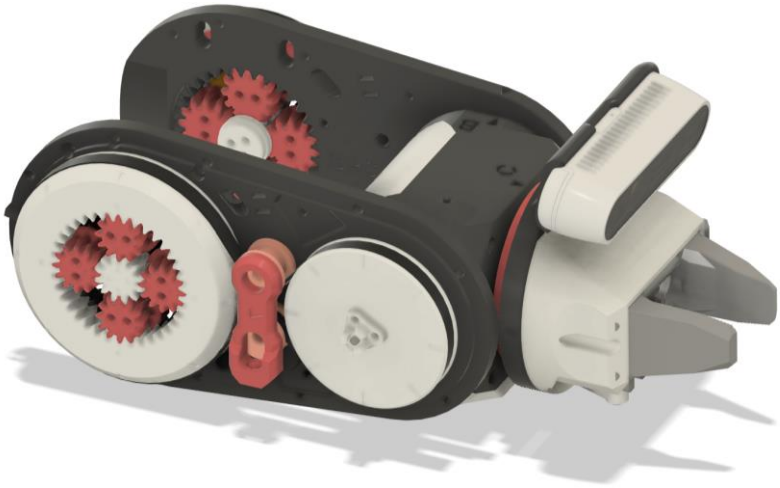
Díky ROS2 struktuře je celý systém modulární a software pro rozpoznávání značek bude v budoucnu jednoduché nahradit zmíněným pokročilým algoritmem pro rozpoznávání obecných objektů.

Kamera byla umístěna v lokální eye-in-hand konfiguraci. Lokální kamera je robustní vůči překážkám, které by mohly zamezit globální kameře pohled na TCP a sledovaný objekt. Zároveň není nutné využívat speciální kameru, jejíž zorné pole by pokrylo celý pracovní prostor manipulátoru. Určení polohy objektu lokální kamerou a následné řízení je preciznější, protože se kamera fyzicky nachází blíže ke sledovanému objektu.

4.4.1 Uchycení kamery

V rámci práce bylo nutné navrhnout způsob upevnění kamery k manipulátoru. Jako výchozí díl pro uchycení byla zvolena základna posledního kloubu. Kamera musí být uchycena k poslednímu kloubu, aby měla stejný počet stupňů volnosti jako má manipulátor. Konkrétně základna byla pro uchycení zvolena z toho důvodu, že se jedná o nejvíce mohutný a stabilní díl ze všech částí posledního kloubu. Uchycení kamery musí

být dostatečně robustní na to, aby nepodléhalo vibracím a deformacím a aby se relativní póza kamery k TCP neměnila. Kamera je orientována se sklonem 10° vzhledem k rovině gripperu, aby dokázala zachytit předměty i v okamžiku uchopení. Kromě základny byl modifikován také kryt gripperu. Výsledné uchycení je zobrazeno v Obr. 26.



Obr. 26: Gripper s kamerou

Kamera je k řídicímu systému připojena přes UBS-C kabel. Bylo tedy nutné navrhnout způsob přivedení datového kabelu ze základy manipulátoru až ke kameře. Kabel vede skrz poslední kloub, čímž se předchází jeho zachycení o okolní konstrukci. Modifikovaná základna posledního kloubu je zobrazena v Obr. 27.

Obr. 27: Modifikovaná základna posledního kloubu

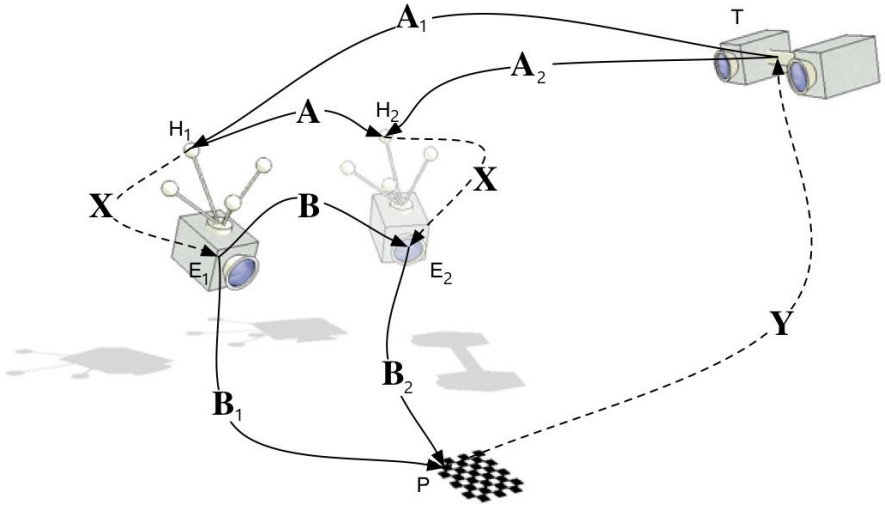
4.4.2 Platforma ViSP Inria

Existuje platforma ViSP (Visual Servoing Platform), která zajišťuje mnoho funkcí týkajících se algoritmů řízení s vizuální zpětnou vazbou. ViSP byl navržen tak, aby byl nezávislý na hardwaru, snadno použitelný, rozšiřitelný a multiplatformní. Umožňuje návrh úloh pro systémy typu eye-in-hand a eye-to-hand pomocí nejtypičtějších vizuálních charakteristik, které se běžně používají v praxi. Nabízí algoritmy zpracování obrazu nebo sledování objektů na základě známých CAD modelů. [28]

4.4.3 Kalibrace eye-in-hand kamery

Pro zaručení funkčnosti všech ostatních algoritmů je nutné nejprve precizně zjistit

relativní pózu mezi kamerou a TCP manipulátoru. Míra přesnosti této transformace ovlivňuje veškerá další měření, která jsou využita k řízení manipulátoru. Pokud během kalibrace vznikne odchylka, nebude vizuální zpětná vazba fungovat korektně. Proces kalibrace eye-in-hand kamery se nazývá Hand-Eye kalibrace.



Obr. 28: Hand-eye kalibrace [29] (upraveno)

Proces kalibrace využívá vytištěnou šachovnici nebo jiný, dobře rozpoznatelný vzor. Ten je v Obr. 28 označen písmenem $\boldsymbol{P}$. V rámci této práce byla pro kalibraci využita značka typu AprilTag, jelikož se používá také pro označení samotných objektů při následném vizuálním řízení. Spolehlivost balíku pro detekci tohoto typu značky již byla otestována a nebylo tedy nutné vytvářet nový detektor speciálně pro šachovnici.

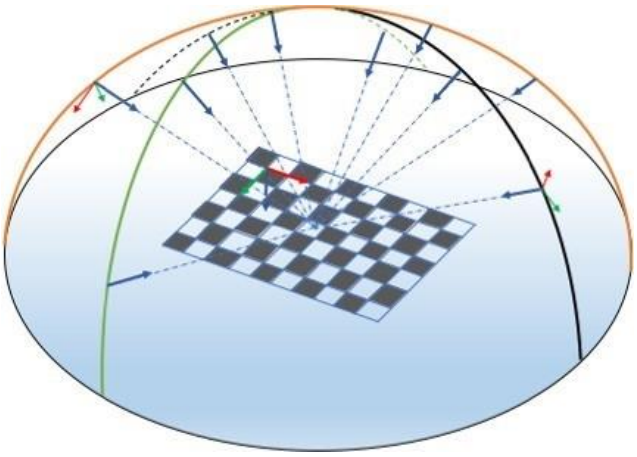
Písmeno $\boldsymbol{T}$ v Obr. 28 značí referenční bod kalibrace, například počátek souřadnicového systému robotu. Aktuální pozice koncového bodu manipulátoru je označena písmenem $\boldsymbol{H}$. Aktuální pozici kamery značí písmeno $\boldsymbol{E}$. Cílem kalibrace je odvodit transformaci, která je označena písmenem $\boldsymbol{X}$ a vyjadřuje relativní pózu mezi koncovým bodem manipulátoru $\boldsymbol{H}$ a kamerou $\boldsymbol{E}$. Póza H je známá díky transformaci získané z přímé kinematiky. Tato transformace je vyjádřena písmeny $\boldsymbol{A_1}$ a $\boldsymbol{A_2}$. Z obrazů pořízených kamerami v pozicích $\boldsymbol{E_1}$ a $\boldsymbol{E}_2$ je možné odvodit transforace $\boldsymbol{B_1}$ a $\boldsymbol{B_2}$ mezi kamerou $\boldsymbol{E}$ a značkou $\boldsymbol{P}$.

Proces kalibrace probíhá následovně. Manipulátor je nastaven do pózy $\boldsymbol{H_1}$. Kamera nasnímá značku a z obrazu je vypočtena póza $\boldsymbol{E_1}$. Následně je manipulátor přesunut do jiné pózy $\boldsymbol{H_2}$. Kamera opět nasnímá značku a je vypočtena póza $\boldsymbol{E_2}$. Dalším krokem je výpočet relativní změny pózy robotu mezi body $\boldsymbol{H_1}$ a $\boldsymbol{H_2}$ a také relativní změny pózy kamery mezi body $\boldsymbol{E_1}$ a $\boldsymbol{E_2}$. Platí vztahy *(4.2)* a *(4.3)*.

$$\boldsymbol{A = H_2 - H_1} \qquad (4.2)$$

$$\boldsymbol{B = E_2 - E_1} \qquad (4.3)$$

Přesnost kalibrace se zvyšuje při opakování popsaného procesu pro několik různých poloh. Kalibraci považujeme za vyhovující, pokud využité polohy manipulátoru pokrývají co největší oblast polokoule nad značkou, jak ukazuje Obr. 29.



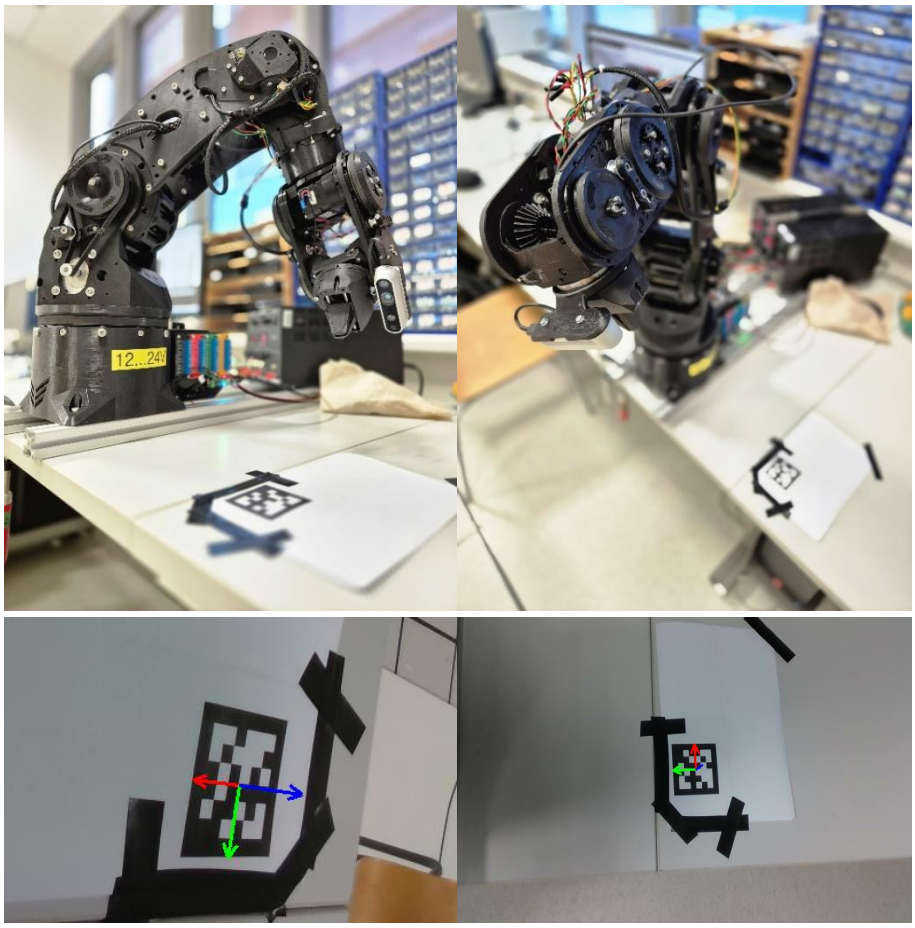
Obr. 29: Vhodné pózy měření při hand-eye kalibraci [30]

Pro matice $\boldsymbol{A}$ a $\boldsymbol{B}$ platí vztah *(4.4)*. Cílem kalibrace je nalézt takovou matici homogenní transformace $\boldsymbol{X}$, pro kterou se obě strany rovnice co nejvíce podobají pro všechny změřené pózy. Je tedy nutné vyřešit úlohu minimalizace popsanou vztahem *(4.5)*. Nalezení takového řešení v rámci této práce zajišťuje nástroj visp-compute-eye-in-hand-calibration z platformy ViSP.

$$\boldsymbol{A} \cdot \boldsymbol{X} = \boldsymbol{X} \cdot \boldsymbol{B} \tag{4.4}$$

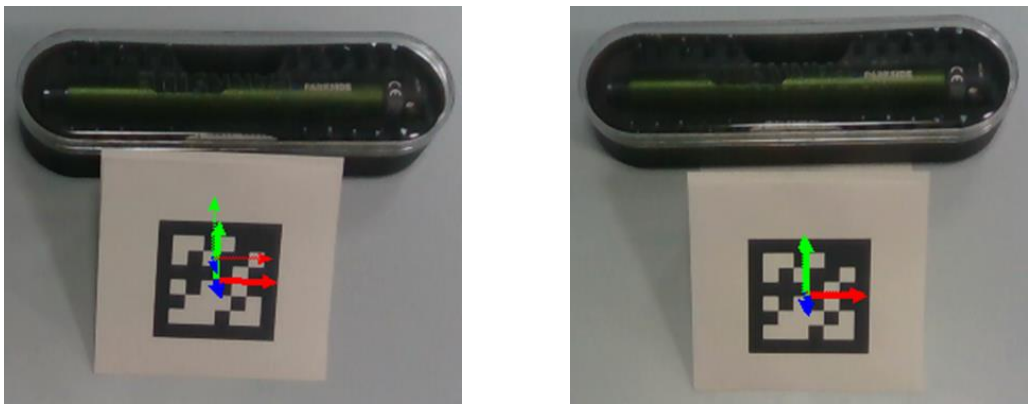
$$\min_{X} \sum_{i=0}^{n} \left|\left|\boldsymbol{A_i X} - \boldsymbol{X B_i}\right|\right| \tag{4.5}$$

Pro účely Hand-Eye kalibrace vznikl nový ROS2 uzel HandEyeCalib. Ten ukládá kalibrační data a jeho integrace do systému je detailněji popsána v kapitole 5.6. V rámci kalibrace proběhlo měření ve více než třiceti rozdílných polohách, které pokryly doporučenou oblast polokoule nad značkou. Průběh kalibrace je zobrazen v Obr. 30.



Obr. 30: Průběh hand-eye kalibrace

Po uložení všech potřebných dat je možné spočítat transformaci $\boldsymbol{p_{TCP} = H_{TC} \cdot p_{cam}}$. O to se stará zmíněný ViSP nástroj visp-compute-eye-in-hand-calibration. Ten přečte všechny uložené soubory, provede popsané výpočty a jako výsledek poskytne transformační matici $\boldsymbol{H_{TC}}$. V Obr. 31 je znázorněna přesnost vizuálního řízení před a po provedení kalibrace. Čárkovanými šipkami je zobrazena požadovaná póza, tučné šipky znázorňují aktuální detekovanou pózu. Před provedením kalibrace byla transformační matice $\boldsymbol{H_{TC}}$ odvozena měřením na CAD modelu.



Obr. 31: Před kalibrací (vlevo), po kalibraci (vpravo)

Je nutné podotknout, že dosažení takto přesné pózy zabralo vlivem slabé konstrukce s omezenou dynamikou několik minut. Při běžném provozu vizuální řízení takové přesnosti nedosahuje, jelikož to konstrukce v rozumném čase nedovolí. Je definováno toleranční pásmo kolem cílové pózy, ve kterém je regulace považována za dokončenou. Tím je sice omezena přesnost řízení, ale celý proces trvá podstatně kratší dobu.

4.4.4 Zpracování dat z hloubkové kamery Intel RealSense

Intel poskytuje ke svým hloubkovým kamerám řady RealSense sadu nástrojů knihoven, dokumentaci a příklady použití. Hloubková data mohou být reprezentována jako mračno bodů, nebo jako vzdálenost každého změřeného bodu zarovnaná na barevné pixely obrazu. Existuje jak základní knihovna s označením librealsense, tak kompletní balíček pro ROS2 realsense-ros. [31]

Integraci hloubkové kamery do systému zajišťuje ROS2 uzel CameraDriver. Jeho detailnímu popisu se věnuje kapitola 5.4.

4.4.5 Implementace vizuálního řízení

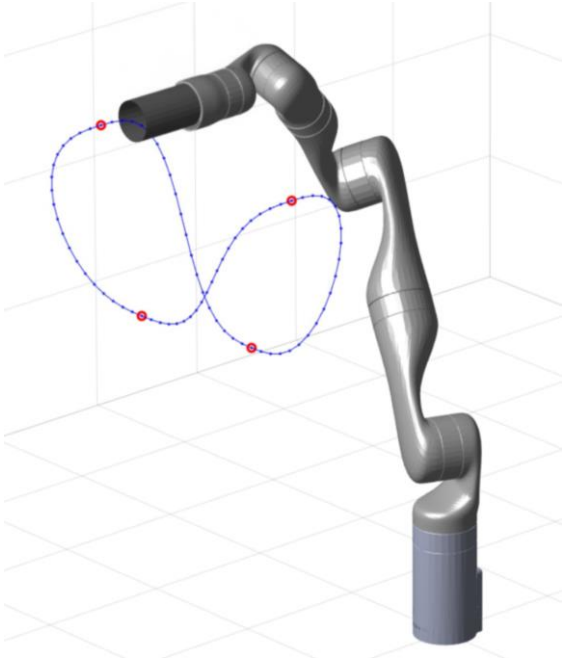
Samotné vizuální řízení metodou PBVS využívá algoritmy z platformy ViSP. Tuto funkcionalitu zajišťuje ROS2 uzel s označením VisualServoing, kterému se detailně věnuje kapitola 5.7.

4.5 Plánování trajektorie

Pro splnění požadovaných úloh musí být manipulátor schopen vykonávat specifické pohyby v kartézském prostoru s důrazem na přesnost provádění lineárních pohybů a dalších definovaných trajektorií. To bez algoritmu pro plánování trajektorie není možné. Tyto algoritmy manipulátoru umožňují interagovat s objekty definovaným způsobem. Robotické rameno díky nim může například uchopit těleso, nebo různá tělesa spojovat, vrtat do nich atd. Specifická aplikace závisí na nástroji, který je na manipulátoru připevněn a zadání dané úlohy.

V robotice se plánovače trajektorie běžně používají pro definování požadovaných pohybů robotu. Specificky u manipulátorů je naplánovaná trajektorie zaměřená na pohyby TCP. Typicky plánovače rozdělujeme do dvou kategorií. Offline a online plánovače.

Offline plánovače počítají celou trajektorii ještě před začátkem samotné operace. Výstupem je tedy kompletní a pevný soubor póz, kterých má robot dosáhnout. Vygenerovaná trajektorie může být globálně optimální vzhledem k různým kritériím (čas, spotřebovaná energie, plynulost pohybů, …). Robot poté danou trajektorii slepě vykonává a nereaguje na změny v prostředí. Tento typ plánovačů je vhodný například v průmyslových výrobních halách, kde je prostředí statické a manipulátor vykonává neustále naprosto stejnou operaci. Funkce offline plánovače trajektorie je ilustrována v Obr. 32. [32]

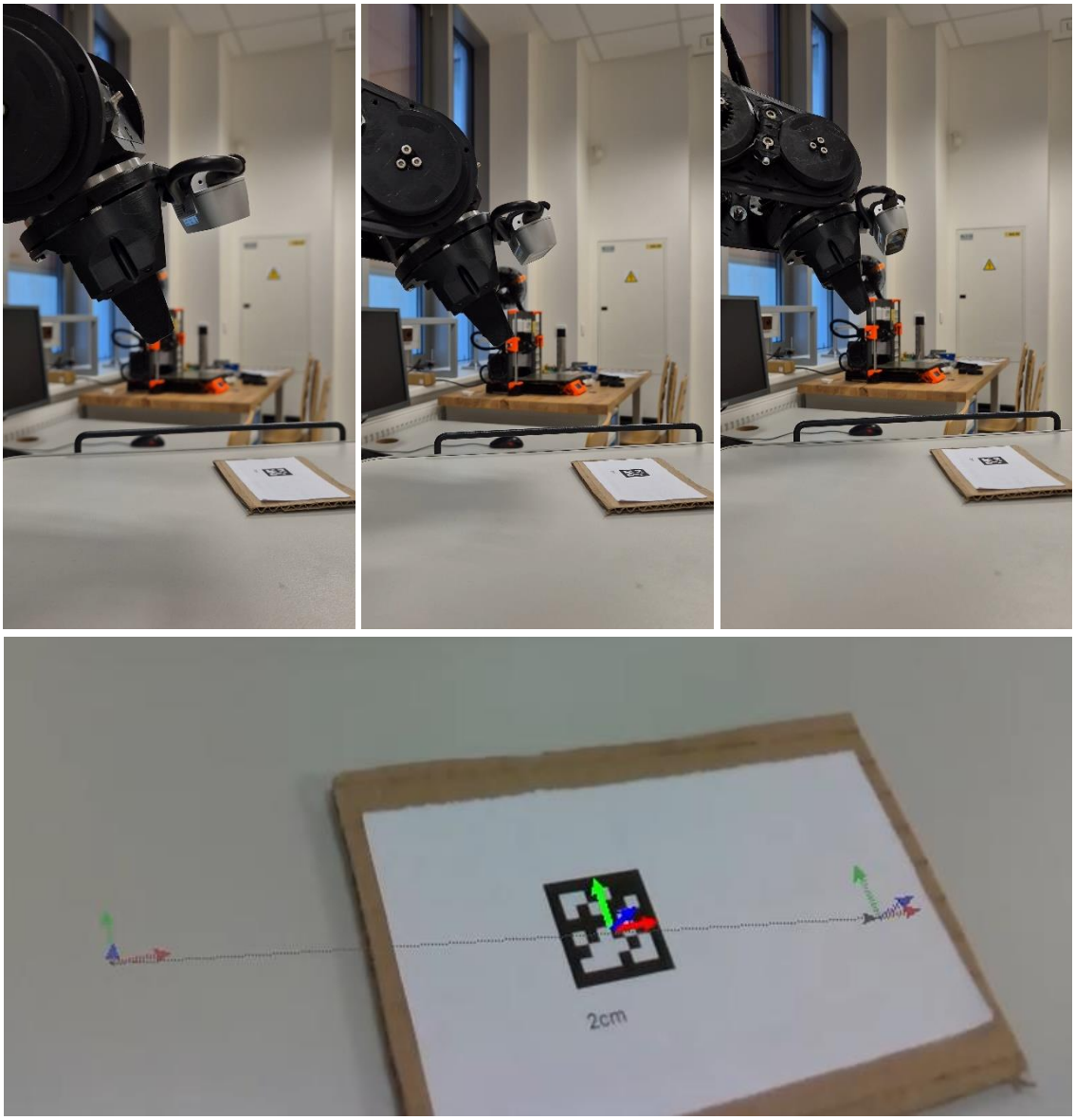


Obr. 32: Offline plánovač trajektorie [33] (upraveno)

Naopak online plánovače trajektorii generují postupně. V každém kroku reagují na změny prostředí a počítají následující pohyby. Tyto plánovače jsou vhodné do dynamického prostředí, kde se stav pracovního prostoru v čase mění. Tento typ plánovače typicky nemá k dispozici kompletní mapu prostředí a vytvořená trajektorie tak může být optimální pouze lokálně. [32]

V této aplikaci je využit offline plánovač trajektorie. Uplatňuje se však nestandardním způsobem. Už nezajišťuje přímo plánování trajektorie koncového bodu manipulátoru v kartézském prostoru, ale spolupracuje s algoritmem vizuálního řízení. Algoritmus PBVS počítá akční zásahy robotu tak, aby byla minimalizována odchylka mezi aktuální a požadovanou pózou sledovaného objektu vzhledem ke kameře. Plánovač trajektorie postupně a plynule upravuje požadovanou pózu podle předem vygenerované trajektorie. Požadovaná póza představuje relativní pohyby mezi objektem a kamerou. Pohyby manipulátoru kontinuálně koriguje PBVS algoritmus.

Díky popsanému přístupu je možné celou trajektorii, která popisuje danou interakci s objektem, detailně naplánovat a odladit předem. Manipulátor je v průběhu vykonávání této trajektorie schopen reagovat na dynamické změny prostředí. Objekt tak v průběhu vykonávání operace nemusí být naprosto stacionární a může se pohybovat. Při precizním provedení vizuálního řízení je tak možné do pohybujícího se sledovaného objektu například vrtat bez hrozby zničení nástroje či znehodnocení obrobku. Stejně tak není nutné daný objekt umístit vždy na naprosto totožné místo. Tato skutečnost je obzvlášť výhodná například při pick-and-place operacích v kolaboraci s člověkem. Stačí, když je daný objekt umístěn kdekoliv uvnitř pracovního prostoru manipulátoru a vizuální řízení zajistí zbytek. Průběh vykonávání trajektorie popsaným způsobem ilustruje Obr. 33.



Obr. 33: Vykonávání offline trajektorie s PBVS

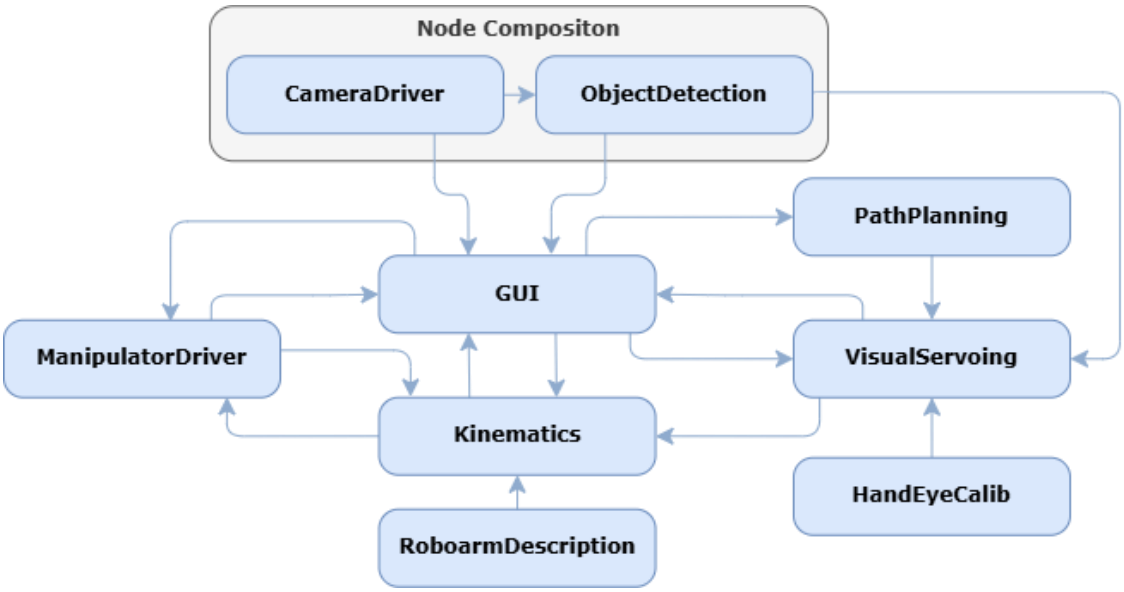
5 VLASTNÍ IMPLEMENTACE – ROS2 NETWORK

V této kapitole bude popsána struktura navrženého systému, jeho rozdělení na jednotlivé moduly a komunikace mezi nimi. V ROS2 jsou tyto moduly označovány jako uzly (anglicky node). Všechny uzly jsou vypsány v Tab. 3.

Tab. 3: Seznam ROS2 uzlů projektu

Název	Funkce	Jazyk
ManipulatorDriver	Řízení pohybů jednotlivých kloubů manipulátoru	Python
Kinematics	Výpočet přímé a inverzní kinematiky	Python
CameraDriver	Sběr dat z kamery	C++
ObjectDetection	Detekce pózy objektu	C++
HandEyeCalib	Hand-eye kalibrace kamery	C++
VisualServoing	Vizuální řízení metodou PBVS	C++
PathPlanning	Plánování trajektorie	Python
Gui	Grafické uživatelské rozhraní	Python
RoboarmDescription	CAD a URDF popis manipulátoru	XML

Modulární struktura vytvořeného systému je zobrazena v Obr. 34 jako zjednodušené schéma sloužící pro ilustraci základního uspořádání uzlů.



Obr. 34: Nákres modulární struktury systému

V textu se předpokládá, že čtenář má základní znalosti týkající se ROS2 architektury. Pojmy jako Publisher, Subscriber atd. tedy nebudou vysvětlovány. V opačném případě doporučujeme navštívit dokumentaci ROS2 [34].

5.1 Datové typy zpráv

V ROS2 není nutné veškeré zprávy definovat ručně. Existuje seznam nativních knihoven, které definují nejčastěji používané zprávy. Specifické datové typy zpráv se v našem systému používají opakovaně, proto je pro ně vyhrazena tato kapitola. V ní jsou veškeré používané typy zpráv popsány, aby se předešlo opakovanému vysvětlování ve zbytku textu.

5.1.1 Sensor_msgs

Tato knihovna [35] definuje zprávy pro nejčastěji používaní senzory a akční členy včetně snímačů polohy a kamer.

První typ zprávy, který je touto knihovnou definován je **JointState.msg**, jejíž strukturu ukazuje Výpis 1. Zpráva obsahuje hlavičku, která zastřešuje identifikátor zprávy, časovou značku a název systému, kterému patří. Dále je přenášeno pole názvů jednotlivých kloubů a tři pole pro pozici, rychlost a sílu. Při přenosu zpráv mohou být některá políčka zanechána nevyplněná. JointState zprávy jsou v naší aplikaci používány pro přenos informací ohledně aktuální polohy a rychlosti všech kloubů. Tyto zprávy jsou taktéž využity při zadávání požadované polohy a rychlosti.

Výpis 1: Struktura JointState.msg

```
std_msgs/Header header
string[] name
float64[] position
float64[] velocity
float64[] effort
```

Dalším typem zprávy, který je využit v naší aplikaci a je definován knihovnou sensor_msgs je **Image.msg**. Jeho strukturu ukazuje Výpis 2. Zpráva opět obsahuje standardní hlavičku. Mezi další atributy patří rozměry obrázku v pixelech. Další položkou je encoding, tedy formát jednotlivých pixelů. Tato položka vychází z knihovny sensor_msgs::image_encoding, která definuje formáty například rgb8, rgba8 aj. Obsahuje také podporu pro formáty z knihovny OpenCV, jako například 8UC3, 8UC4 atd. Parametr is_bigendian udává byte order a uplatní se u obrazových formátů, které jsou reprezentovány více než 8 bity, což v této aplikaci nenastává. Položka step udává počet bajtů na jeden řádek a pole data už potom představuje samotná přenášená data.

Výpis 2: Struktura Image.msg

```
std_msgs/msg/Header header
uint32 height
uint32 width
string encoding
uint8 is_bigendian
uint32 step
uint8[] data
```

Poslední typ zprávy z této knihovny je **CameraInfo.msg**. Její strukturu ukazuje Výpis 3. Kalibrační data, která poskytuje použitá kamera představují model zkreslení s jeho parametry D a matici intrinsických parametrů K. Dále pak rozměry obrazu. Zbylé položky této zprávy tak nejsou využity.

Výpis 3: Struktura CameraInfo.msg

```
std_msgs/Header header
uint32 height
uint32 width
string distortion_model
float64[] D
float64[9] K
float64[9] R
float64[12] p
uint32 binning_x
uint32 binning_y
sensor msgs/RegionOfInterest roi
```

5.1.2 Geometry_msgs

Knihovna geometry_msgs [36] definuje zprávy pro běžně používaná geometrická primitiva, jako například bod, vektor, póza atd.

Zpráva **PoseStamped.msg** slouží pro přenos informací o póze souřadnicového systému. Kromě standardní hlavičky se tedy skládá ze složek polohy a orientace. Poloha je reprezentována jako bod se třemi složkami. Orientace je poté reprezentována kvaternionem. Kompletní strukturu zprávy zobrazuje Výpis 4.

Výpis 4: Struktura PoseStamped.msg

```
std_msgs/Header header
geometry_msgs/Pose pose
  geometry_msgs/point position
    float64 x
    float64 y
    float64 z
  geometry_msgs/Quaternion orientation
    float64 x
    float64 y
    float64 z
    float64 w
```

Druhý typ zprávy týkající se geometrie je **TwistStamped.msg**. Podobně jako PoseStamped slouží pro reprezentaci souřadnicového systému v prostoru. Na rozdíl od PoseStamped ale TwistStamped přenáší data o twistu souř. systému, tedy o jeho lineárních a rotačních rychlostech. Jeho strukturu zobrazuje Výpis 5.

Výpis 5: Struktura PoseStamped.msg

```
std_msgs/Header header
geometry_msgs/Twist twist
  geometry_msgs/vector3 linear
    float64 x
    float64 y
    float64 z
  geometry_msgs/Vector3 angular
    float64 x
    float64 y
    float64 z
```

5.1.3 Nav_msgs

Knihovna nav_msgs [37] definuje nejčastěji používané zprávy spojené s navigací. Naše aplikace využívá zprávu **GetPlan.srv**, která slouží pro komunikaci mezi service klientem a server. Její strukturu zobrazuje Výpis 6. Jako požadavek jsou předány dva PoseStamped argumenty. Položka tolerance v naší aplikaci není nevyužita. Odpověď potom zahrnuje naplánovanou trajektorii, která je reprezentována polem póz.

Výpis 6: Struktura GetPlan.srv

```
#Request
geometry_msgs/PoseStamped start
geometry_msgs/posestamped goal
float32 tolerance
---
#Response
nav_msgs/Path plan
  std_msgs/Header header
  geometry_msgs/PoseStamped[] poses
```

5.1.4 Std_srvs

Knihovna std_srvs [38] definuje základní service zprávu typu **Trigger.srv**. Při využití této zprávy service klient nepředává žádná data, jen jednoduše vyvolává service server rutinu. Jako odpověď je předána proměnná typu boolean, která reprezentuje úspěšné/neúspěšné dokončení požadavku, případně slovní hláška. Strukturu této zprávy zachycuje Výpis 7.

Výpis 7: Struktura Trigger.srv

```
#Request

---
#Response
boolean success
string message
```

5.1.5 Vlastní zprávy

Přestože ROS poskytuje nepřeberné množství nejčastěji používaných zpráv, pro některé specifické operace bylo v rámci našeho systému nutné definovat vlastní zprávy. Ve

zbytku textu jsou vlastní typy zpráv označeny hvězdičkou.

První vlastní zprávou je **TriggerAxis.srv**. Jedná se o service zprávu, která jako požadavek předává index kloubu a jako odpověď poskytuje boolean proměnnou, která značí úspěch či neúspěch operace. Tato zpráva je využita při inicializaci a nulování jednotlivých kloubů. Je definována v balíku manipulator_servo_driver_interfaces a její strukturu zobrazuje Výpis 8.

Výpis 8: Struktura TriggerAxis.srv

```
#Request
int8 index
---
#Response
bool success
```

Typ zprávy **ChangeMode.srv** slouží pro změnu režimu řízení manipulátoru. Je taktéž definován v balíku manipulator_servo_driver_interfaces. Interface zobrazuje Výpis 9.

Výpis 9: Struktura ChangeMode.srv

```
#Request
uint8 mode
---
#Response
bool success
```

Zpráva **CameraParams.srv** definovaná v balíku camera_driver_interfaces a slouží pro vyžádání parametrů kamery. Při požadavku se nepředávají žádná data. Odpověď představuje typ CameraInfo z knihovny sensor_msgs (viz kapitola 5.1.1). Parametry kamery jsou čteny jen při spuštění, není tak nutné je publikovat neustále. Strukturu zprávy zobrazuje Výpis 10.

Výpis 10: Struktura CameraParams.srv

```
#Request

---
#Response
Sensor_msgs/cameralnfo camera info
```

Zpráva **ExecutePath.action** slouží jako interface mezi klientem a server ROS2 action. Je definovaná v balíku visual_servoing_interfaces a používá se pro vykonávání naplánované trajektorie při vizuálním řízení. Jako požadavek je předána trajektorie path, která je reprezentována jako pole PoseStamped. Action server tuto akci vykonává a průběžně odesílá feedback step. Po dokončení, případně selhání trajektorie server odesílá výsledek status. Strukturu interface zobrazuje Výpis 11.

Výpis 11: Struktura ExecutePath.action

```
# Request
nav_msgs/Path path
---
# Result
uint8 status
---
# Feedback
uint32 step
```

5.2 Driver manipulátoru

Uzel ManipulatorDriver zajišťuje rozhraní mezi logickou vrstvou zbytku ROS2 Network a fyzickou vrstvou v podobě šestice servopohonů. Ty byly popsány v kapitole 1.1. Driver pro komunikaci se servy využívá CAN sběrnici.

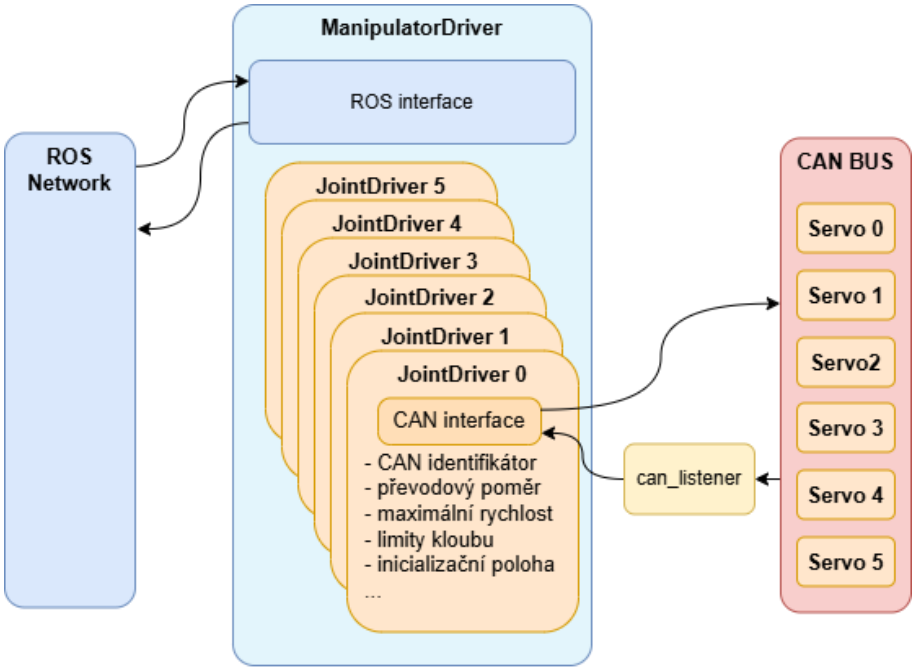
Pro sestavování CAN zpráv a jejich základní validaci byly použity části kódu z projektu mks-servo-can [39]. Tyto jednoduché funkce vychází z dokumentace serv [3], byly však převzaty, aby se předešlo zbytečnému opisování triviálního kódu. Veškerá zbylá funkcionalita byla implementována samostatně.

Komunikační rozhraní uzlu ManipulatorDriver v rámci ROS2 Network je popsáno Tab. 4. Topicy využívají standardní ROS2 datové typy. Výjimkou jsou zprávy označené hvězdičkou. Ty jsou nově definované.

Tab. 4: Komunikační rozhraní uzlu ManipulatorDriver

Role	Topic	Komunikační partner	Datový typ	Popis
Subscriber	manipulator/ set_joints_position	GUI, Kinematics	JointState	Příjem požadavku na natočení kloubů
Subscriber	manipulator/ set_joints_velocity	GUI, Kinematics	JointState	Příjem požadavku na rychlosti kloubů
Publisher	manipulator/ state_joints_position	GUI, Kinematics	JointState	Aktuální natočení kloubů
Publisher	manipulator/ state_joints_velocity	GUI, Kinematics	JointState	Aktuální rychlosti kloubů
Service server	manipulator/ change_mode	GUI	ChangeMode*	Přepnutí režimu řízení
Service server	manipulator/ home_single_joint	GUI	HomeAxis*	Inicializace kloubu
Service server	manipulator/ zero_single_joint	GUI	ResetAxis*	Nastavení kloubu do domovské pozice

Finální architektura uzlu ManipolatorDriver je naznačena v Obr. 35. Architektura asynchronního driveru je navržena následovně. Po spuštění programu je vytvořena třída ManipulatorDriver jako potomek třídy rclpy.Node. Tento uzel slouží pro komunikaci v rámci ROS2 Network. Během inicializace vyhradí samostatné vlákno, na kterém je spuštěn asyncio executor, odpovědný za správu asynchronních akcí nad sběrnicí CAN. Práce s knihovnou asyncio a vícevláknové programování je detailně popsáno v kapitole 5.2.3. Pro každé servo je vytvořena instance třídy JointDriver s unikátním identifikátorem, která zajišťuje asynchronní komunikaci s jeho fyzickým protějškem. Identifikátor každého JointDriveru je totožný s HW identifikátorem odpovídajícího serva.



Obr. 35: Základní schéma architektury uzlu ManipulatorDriver

5.2.1 Počáteční stav driveru

Driver manipulátoru byl jediný kus softwaru, který byl na začátku práce převzat společně s hardwarem manipulátoru. Původní implementace této části systému byla považována za dokončenou a funkční. Předpokládalo se, že nebude vyžadovat další zásahy. Během následné integrace se však ukázalo, že kód driveru nesplňuje požadované funkční nároky, což si vyžádalo jeho kompletní přepracování s cílem zajistit správnou funkčnost celého systému. Výchozí stav driveru je popsán v následující kapitole 5.2.2.

5.2.2 Synchronní přístup ke sběrnici CAN

První pokusy o zprovoznění komunikace se servy stály na čistě synchronní architektuře, jak ukazuje zmíněný projekt mks-servo-can [39]. Serva využívají následující formát komunikace. Software JointDriver odešle požadavek do hardwarového driveru MKS-SERVO. Ten provede danou akci (např. změření polohy, nastavení rychlosti atd.). Posléze odesílá odpověď, kterou JointDriver dekóduje. Při čistě synchronní komunikaci je během čekání na odpověď vykonávání programu blokováno. V základu bylo čekání na odpověď nastaveno na 100 ms. To při synchronní komunikaci se šesti servy vedlo ke zpoždění 600 ms na jeden dotaz. Při cyklickém dotazování na aktuální polohu a rychlost tak docházelo k zablokování programu na minimálně 1,2 sekundy v každém cyklu. V případě, že se pouze četla poloha a rychlost. Některé příkazy, jako například nastavení polohy, posílají odpovědi dvě. První při spuštění pohybu a druhou při úspěšném najetí na požadovanou polohu. Vlivem synchronního přístupu ke sběrnici bylo nutné buď zablokovat vykonávání programu po celou dobu provádění příkazu, nebo druhou odpověď ignorovat. Jak je z popisu zřejmé, tento přístup nebyl vyhovující.

Další testovaná varianta byla stále synchronní, komunikace se servy však neprobíhala sekvenčně jedno po druhém, ale požadavek se odeslal do všech serv současně. Tím zpoždění reakce kleslo z 600 ms za jeden dotaz na 100 ms. Hromadné odesílání požadavků do všech serv a následné synchronní čekání sice potlačilo problém s dlouhou odezvou, ale nebylo dostatečně robustní vůči sporadickým zpožděním nebo úplným ztrátám odpovědí.

Ve finální aplikaci je zaveden asynchronní přístup ke sběrnici, který je postaven na knihovně asyncio.

5.2.3 Asynchronní přístup ke sběrnici CAN

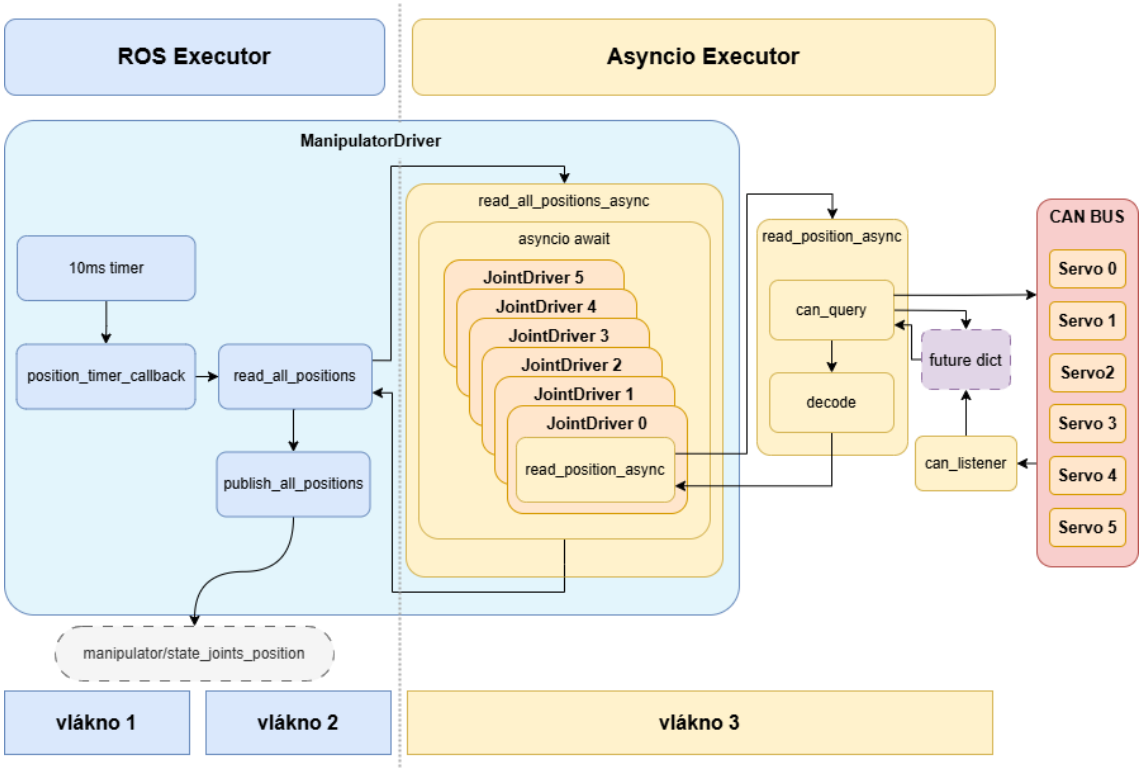
V ROS2 je vykonávání úkolů řízeno pomocí tzv. executoru. Ten vytváří tzv. wait-set. Jedná se o souhrn všech aktuálně aktivních entit, které mohou vyvolávat události. Takové entity jsou například časovače, subscribeři atd. Události, které tyto entity vyvolávají jsou označovány jako callback funkce. Executor cyklicky kontroluje, zda nějaká entita vyvolala událost a podle toho spravuje vykonávání callback funkcí. Tento okamžik kontroly se nazývá polling point. Doba mezi dvěma polling points se nazývá processing window. V tomto časovém okně executor zpracovává zaznamenané callbacky. [40] [41]

Callback funkce by měly být poměrně rychlé, aby neblokovaly vlákno ROS2 executoru. Odpovědi z CAN sběrnice však mohou dorazit klidně po několika desítkách vteřin v případě požadavku o nastavení polohy. Proto byl nasazen asynchronní přístup ke sběrnici pomocí knihovny asyncio. Asyncio implementuje vlastní executor označovaný jako „event_loop“. Asyncio exekutor však nedokáže s ROS2 exekutorem spolupracovat. Nedokážou si efektivně rozdělit jedno vlákno procesu. Proto je nutné, aby asyncio executor běžel na vlákně, které je oddělené od toho, na kterém běží ROS.

Příklad zpracování callbacku na zjištění aktuální polohy všech kloubů je ilustrován na Obr. 36. Tento callback je vyvolán každých 20 ms. Při spuštění aktivuje nové vlákno, na kterém vyvolá obslužnou rutinu, a okamžitě se opět ukončí. Díky tomu je dodržen předpoklad, že callbacky by měly být rychlé operace, které neblokují ROS2 executor. Na novém vlákně je tedy spuštěna, stále ještě synchronní, funkce read_all_positions. Tato funkce předává asyncio executoru požadavek na asynchronní vykonání funkce read_all_positions_async. Následně čeká v blokujícím čekání, dokud nedostane všechny výsledky. Zde ale blokující čekání nevadí, protože probíhá na odděleném vlákně a neblokuje ROS2. Asyncio executor mezitím spustí požadovanou funkci.

Asynchronní funkce jsou označovány jako korutiny (anglicky coroutines). Uvnitř korutiny read_all_positions_async je vytvořena sada šesti úkolů read_position_async. Pro každý JointDriver jedna. Paralelní vykonání těchto úkolů řídí asyncio executor. Korutina read_position_async si vyžádá korutinu can_query. Jedná se o nejnižší korutinu při asynchronním přístupu ke CAN sběrnici. Uvnitř can_query se sestaví odpovídající CAN, která se poté odešle. Do sdíleného slovníku can_responses_dict se vloží nová future s klíčem obsahujícím identifikátor serva a kód zprávy. Future je objekt, který představuje

přislib budoucích hodnot. Can_query poté asynchronně čeká na naplnění své future. Během toho jsou vykonávány ostatní operace. CAN sběrnice přenese zprávu do serva, které reaguje zasláním odpovědi. Veškeré odpovědi sbírá can_listener, který běží v nekonečné smyčce. Podle identifikátoru serva a kódu zprávy v obdržené odpovědi najde odpovídající future ve slovníku a naplní ji obdrženými hodnotami. Na to reaguje asyncio executor tím, že probudí čekající korutinu can_query, která hodnoty z future přečte, odstraní ze slovníku a předá jako návratovou hodnotu do funkce read_position_async, která can_query zavolala. Data jsou následně dekódována z hexadecimálního čísla na float a odeslána do nadřazené korutiny read_all_positions_async. Když tato korutina obdrží odpovědi ze všech JointDriverů, odešle hodnoty jako pole floatů do synchronní funkce read_all_positions, která z nich sestaví JointState zprávu a pomocí publishera odešle do ROS2 Network na topicu manipulator/state_joints_position.



Obr. 36: Příklad zjištění polohy všech kloubů

Stejným stylem se zpracovávají veškeré zprávy. Knihovna threading zajišťuje to, že se může vykonávat několik typů příkazů současně. Knihovna asyncio zase umožňuje asynchronní komunikaci se všemi servy najednou. Jelikož je driver vícevláknový, je nutné předcházet race conditions a veškeré sdílené proměnné zamykat pomocí threading.Lock().

5.2.4 Neférové plánování callbacků

Pro účely tohoto textu budeme uvažovat pouze dva základní typy ROS2 executorů. Jednovláknový executor (anglicky single-threaded executor – dále jen STE) a vícevláknový executor (anglicky multi-threaded executor – dále jen MTE).

Požadavky na přečtení aktuálního natočení a rychlosti všech serv vyvolávají dva samostatné timery, které jsou nastaveny na 20ms periodu. Uzel ManipulatorDriver nejprve využíval MTE a callbacky rozděloval do samostatných MutuallyExclusiveCallbackGroups. To teoreticky znamená, že se callbacky mezi odlišnými skupinami mohou vykonávat paralelně, ale callbacky uvnitř jedné skupiny ne. Callbacky jsou tedy paralelní, ale ne reentrantní. Což jsou přesně požadavky na callbacky v této aplikaci. Callbacky timerů pro čtení stavu se měly vykonávat paralelně, nebo alespoň na střídačku. Docházelo však k tomu, že několik cyklů náhodně převažoval jeden callback, poté zase druhý, jak je vidět ve výpisu v Obr. 37.

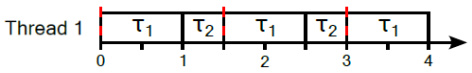
```
[1743705102.589932637]: Velocity: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
[1743705102.602147085]: Encoder values: [0.0, 2.5566346464760684e-06, 0.0, 0.0, 0.0, -0.0]
[1743705102.609146080]: Velocity: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
[1743705102.625460011]: Velocity: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
[1743705102.643922554]: Velocity: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
[1743705102.658473592]: Velocity: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
[1743705102.674760022]: Velocity: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
[1743705102.684246181]: Encoder values: [2.8198176247897817e-05, -0.0, 0.0, 0.0, 0.0, -0.0]
[1743705102.691945522]: Velocity: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
[1743705102.707419826]: Velocity: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
[1743705102.724832458]: Velocity: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
[1743705102.740688061]: Velocity: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
[1743705102.761054733]: Velocity: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
[1743705102.777623751]: Encoder values: [0.0, 2.5566346464760684e-06, 0.0, 0.0, 0.0, -0.0]
[1743705102.781309274]: Velocity: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
[1743705102.795895243]: Velocity: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
[1743705102.810747870]: Velocity: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
[1743705102.828231376]: Velocity: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
[1743705102.843845138]: Velocity: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
[1743705102.860792544]: Encoder values: [2.8198176247897817e-05, -0.0, 0.0, 0.0, 0.0, -0.0]
[1743705102.863574104]: Velocity: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
[1743705102.879062468]: Velocity: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
[1743705102.894326681]: Velocity: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
[1743705102.908292541]: Velocity: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
[1743705102.924683553]: Velocity: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
[1743705102.942423999]: Velocity: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
[1743705102.950524531]: Encoder values: [0.0, 2.5566346464760684e-06, 0.0, 0.0, 0.0, -0.0]
[1743705102.960784319]: Velocity: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
[1743705102.977113148]: Velocity: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
[1743705102.991879468]: Velocity: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
[1743705103.007453805]: Velocity: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
[1743705103.025791282]: Velocity: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
[1743705103.039477614]: Encoder values: [2.8198176247897817e-05, -0.0, 0.0, 0.0, 0.0, -0.0]
[1743705103.126947221]: Encoder values: [2.8198176247897817e-05, 2.5566346464760684e-06, 0.0, 0.0, 0.0, -0.0]
[1743705103.217250084]: Encoder values: [0.0, -0.0, 0.0, 0.0, 0.0, -0.0]
[1743705103.307959594]: Encoder values: [0.0, -0.0, 0.0, 0.0, 0.0, -0.0]
[1743705103.397593686]: Encoder values: [2.8198176247897817e-05, -0.0, 0.0, 0.0, 0.0, -0.0]
[1743705103.488411030]: Encoder values: [0.0, 2.5566346464760684e-06, 0.0, 0.0, 0.0, -0.0]
```

Obr. 37: Výpis z MTE ManipulatorDriveru

Jak je zmíněno na začátku kapitoly 5.2.3, ROS2 executor zajišťuje vykonávání callback funkcí podle událostí entit z wait-setu v okamžicích zvaných polling points. V každém polling point okamžiku executor posbírá všechny vyvolané události a rozdělí je mezi volná vlákna. STE má samozřejmě volné vlákno jedno. **Další polling point nastane, když se kterékoliv vlákno stane nečinným**, tedy když zpracovalo všechny přiřazené události.

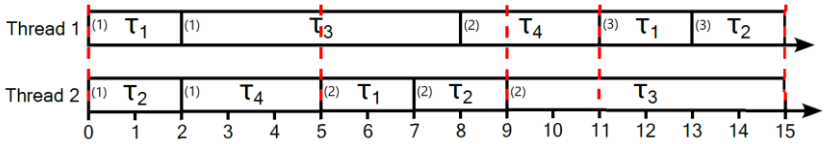
U STE je to jednoduché. Polling je vyvolán po zpracování všech vyvolaných callbacků. Obr. 38 ukazuje příklad, kde je využit STE, který vykonává callbacky dvou timerů T_1 a T_2. Oba timery mají nastavenou periodu 1 s. Doba τ_i značí dobu vykonávání callbacku daného timeru. Polling points jsou zde vyznačeny červenou čárou.

Můžeme pozorovat, že vlákno je přetížené. T_1 i T_2 mají nastavenou periodu 1 s, přesto jsou kvůli dlouhé době vykonání τ opět volány až po době 1,5 s. Z principu STE je však zachováno férové plánování. Callbacky obou timerů se vykonávají sice později, ale ve shodném cyklu. [40]



Obr. 38: STE plánování [40] (upraveno)

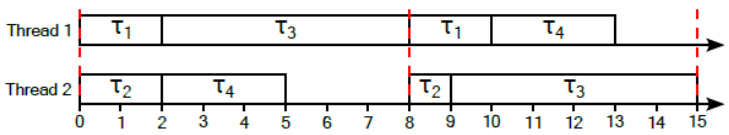
MTE rozděluje události mezi více vláken. Stále platí, že další polling se vyvolá v momentě, kdy **kterékoliv vlákno** dokončí všechny přiřazené události. U MTE má tato skutečnost ale rozdílné dopady. V příkladu na Obr. 39 MTE rozděluje callbacky čtyř timerů mezi dvě vlákna. **Všechny timery** mají zadanou periodu **pět vteřin**. V prvním cyklu MTE přiřadil callbacky T_1 a T_3 prvnímu vláknu a T_2 a T_4 vláknu druhému. Druhé vlákno dokončí přiřazené callbacky v čase 5 s a vyvolává polling point. MTE opět rozdělí aktuální callbacky na své vlákna. [40]



Obr. 39: MTE ASAP plánování [40] (upraveno)

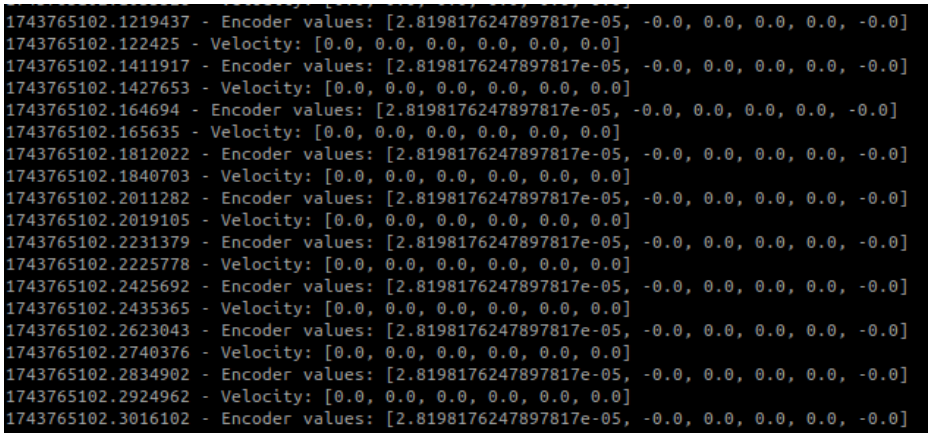
Můžeme si všimnout situace, že všechny timery mají nastavenou totožnou periodu 5 s., přesto v čase 6 s běží T_1 již podruhé, ale T_3 se stále nachází ve svém prvním běhu. To samé např. v čase 14 s., kdy $\boldsymbol{T_1}$ **již dokončil svůj 3. běh a** $\boldsymbol{T_3}$ **stále vykonává běh 2.** Právě tato situace bývá označována za **unfair scheduling**, tedy neférové plánování. Vychází z podstaty implementace MTE v ROS2.

Bylo navrženo řešení [40], kdy polling není vyvolán ihned po uvolnění některého z vláken (as soon as possible – ASAP), jak ukazuje Obr. 39, ale až po uvolnění všech vláken (as late as possible – ALAP). Druhý přístup je znázorněn v Obr. 40. ALAP polling zajistí fair planning. To ale způsobí neefektivní využití vláken procesu a sníží propustnost uzlu. ROS2 neumožňuje volbu mezi zmíněnými přístupy MTE plánování a už od své alpha verze v roce 2015 stále poskytuje výhradně ASAP polling [40].



Obr. 40: MTE ALAP plánování [40]

Právě tato vlastnost MTE způsobila nerovnoměrné vyvolávání timer callbacků pro čtení aktuální polohy a rychlosti v Obr. 37. a vedla k nasazení architektury popsané v kapitole 5.2.3, tedy využití STE a vlastní správa vícevláknového vykonávání callbacků. Zprovozněním této architektury došlo k plně paralelní asynchronní komunikaci všech signálů a férovému plánování, jak je zřetelné z Obr. 41.

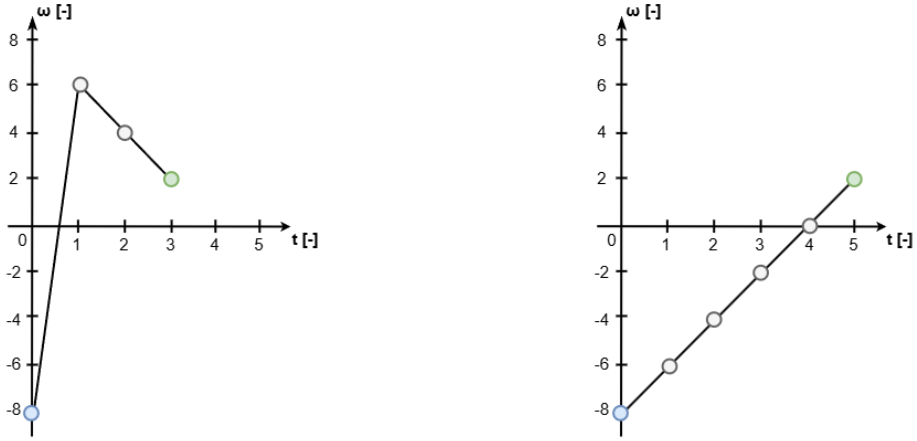


Obr. 41: STE + threading výpis

Z uvedeného výpisu je také možné pozorovat, že odezva na čtení polohy a rychlosti je přibližně 20 ms. Tato odezva je navíc konstantní s počtem souběžných zpráv. Zatímco původní driver při současném čtení polohy a rychlosti a zapisování požadavku o nastavení rychlosti dosahoval zpoždění 1,8 s, implementovaný asynchronní driver si stále drží odezvu 20 ms. **V rámci této práce tedy došlo ke zrychlení komunikace blížící se stonásobku**.

5.2.5 Rampový generátor rychlosti

Hardwarové drivery serv MKS-SERVO-42D a 57D podle dokumentace umožňují postupnou změnu rychlosti s definovaným zrychlením. To vede k plynulosti pohybů a omezení rázů v systému. Vzhledem ke stavu konstrukce manipulátoru se jedná o zásadní schopnost driveru. Během testování se bohužel ukázalo, že uvedený rampový generátor nefunguje podle předpokladů. Absolutní hodnotu rychlosti a směr otáčení řídí HW driver odděleně, jak ukazuje Obr. 42 vlevo. Například při požadavku o změnu rychlosti z -8 na +2 driver nejprve přepne směr otáčení a až poté řeší absolutní hodnotu, což představuje naprosto nevhodné chování. Bylo tedy nutné navrhnout vlastní rampový generátor, který rychlost mění postupně a podle očekávání.



Obr. 42: Rampový generátor driveru. Vlevo původní, vpravo vlastní.

Původní generátor na HW driveru představoval výhodu v tom, že stačilo poslat jeden příkaz s cílovou rychlostí. Nově navržený generátor je však softwarový a je tedy nutné přes CAN opakovaně odesílat plynule měnící se rychlosti. Díky nasazení asynchronního driveru (viz 5.2.3), který snížil odezvu a zavedl paralelní komunikaci tato skutečnost nepředstavuje žádný výrazný problém.

5.2.6 Konfigurace CAN sběrnice

Jak již bylo zmíněno, ManipulatorDriver pro komunikaci se servy využívá komunikaci po CAN sběrnici, jejíchž parametry bylo nutné konfigurovat. HW drivery MKS-SERVO umožňují nastavení jedné ze čtyř podporovaných přenosových rychlostí. Těmi jsou 125 kbps, 250 kbps, 500 kbps a 1 Mbps. Bylo nutné zvolit takovou přenosovou rychlost, aby byla propustnost sběrnice co nejvyšší, ale zároveň aby nedocházelo k jejímu přehlcení.

Nastavená přenosová rychlost vychází z faktu, že rámce CAN zprávy představuje 8 bajtů dat. Většina dotazů se skládá ze dvou zpráv. Žádosti a odpovědi. To tedy odpovídá přenosu 16 bajtů dat na jeden dotaz. Všechny kloubové pohony sdílí společnou sběrnici a dotazy pro všechny serva se odesílají téměř současně. Kompletní dotaz na všechny klouby tak představuje přenos 100 bajtů dat. Maximální přenosová rychlost sběrnice 1 Mbps odpovídá 125 kB/s. Sběrnice tak dokáže pojmout maximálně 1250 dotazů za vteřinu. Komunikace prakticky obsahuje ještě protokolový overhead, takže vypočtené propustnosti nedosáhneme. Jedná se o přibližné výpočty. Propustnost sběrnice omezuje maximální frekvenci dotazů na čtení aktuální polohy a rychlosti kloubů.

Pro výpočet kinematiky je nutné znát aktuální úhly natočení všech kloubů. Tato data tak chceme vyčítat s co nejvyšší frekvencí. Při nastavení obnovovací periody této informace na 1 ms je generováno 1000 dotazů každou vteřinu. Vzhledem k tomu, že sběrnice dokáže ve vteřině pojmout méně než 1250 dotazů, tato obnovovací frekvence je příliš vysoká. Bylo tedy nutné nastavit periodu časovače na 10 ms, aby mohla sběrnice

zpracovávat i jiné typy dotazů než čtení polohy. Maximální přenosová rychlost sběrnice tak omezuje včasnost výpočtů kinematiky. Toto omezení však není zásadní vzhledem ke stavu konstrukce, která zásadně ovlivňuje dynamiku celého systému a je popsána v kapitole 1.3.

Provedení CAN sběrnice není kompletně spolehlivé. Poměrně často dochází ke ztrátě zpráv. To můžeme pozorovat z výpisu nástroje candump, který monitoruje komunikaci na úrovni linkové vrstvy ISO-OSI modelu. Zaznamenává tedy čisté nezpracované rámce přenášené po CAN sběrnici. Tento výpis je zobrazen v Obr. 43. Na základě dokumentace MKS-SERVO byla data dekódována. Význam jednotlivých zpráv je ve výpisu barevně rozlišen. Můžeme pozorovat, že jsme v uvedeném výpisu neobdrželi odpověď od 6. serva. Čekání na odpověď přesáhlo stanovenou hranici a bylo přerušeno a nedostali jsme platná data. Během testování se ukázalo, že tato situace je poměrně častá. Proto byla ošetřena takovým způsobem, že se uchovávají data z předchozího cyklu. Při ztrátě dat se v ManipulatorDriveru tato neplatná hodnota nahradí hodnotou předchozí, která je stará 10 ms. Je nutné podotknout, že neplatí, že by chybně odpovídalo výhradně 6. servo, které je poslední v řetězci. Prakticky přicházíme o zprávy z různých serv a různých dotazů.

Obr. 43: Výpis nástroje candump

Bez rozsáhlejší analýzy není možné zjistit, zda dochází ke ztrátě zpráv na sběrnici CAN, nebo chyba vzniká z MKS-SERVO driveru. Taková analýza však překračuje rozsah této práce.

5.3 Kinematika

Výpočet kinematiky zajišťuje balík Kinematics. Jeho úkolem je počítat přímou úlohu kinematiky (FK), inverzní úlohu kinematiky pózy (IK_pos) a inverzní úlohu kinematiky rychlosti (IK_vel). Tyto výpočty zajišťuje knihovna KDL, která byla ale doplněna řadou vlastních funkcí. Komunikační rozhraní tohoto uzlu je představeno v Tab. 5.

Tab. 5: Komunikační rozhraní uzlu Kinematics

Role	Topic	Komunikační partner	Datový typ	Popis
Subscriber	manipulator/ state_joints_position	ManipulatorDriver	JointState	Aktuální natočení kloubů
Subscriber	kinematics/ set_pose	GUI	PoseStamped	Požadovaná póza
Subscriber	kinematics/ set_velocity	GUI, VisualServoing	TwistStamped	Požadovaná rychlost
Publisher	kinematics/ state_pose	GUI	PoseStamped	Aktuální póza =výsledek FK
Publisher	manipulator/ set_joints_position	ManipulatorDriver	JointState	Požadované natočení kloubů = výsledek IK_pos
Publisher	manipulator/ set_joints_velocity	ManipulatorDriver	JointState	Požadované rychlosti kloubů = výsledek IK_vel
Publisher	joint_states	RViz	JointState	Vizualizace aktuální pózy

5.3.1 Přímá kinematika

Výpočet přímé kinematiky představuje přímočarý postup, jak popisuje kapitola 3.1. Pro odvození pózy TCP se využit solver KDL:: ChainFkSolverPos_recursive. Tato funkce třída implementuje výpočty popsané v kapitole 3.1. Slovo „recursive“ v názvu značí způsob výpočtu. Matice H_{BE} není vypočtena najednou, jak ukazuje rovnice *(4.3)*, ale solver postupně prochází kinematický řetězec od báze po TCP a skládá jednotlivé matice homogenní transformace.

5.3.2 Inverzní kinematika – rychlost TCP

Pro vyvíjenou aplikaci je zásadní, aby bylo možné řídit okamžité lineární a rotační rychlosti TCP v kartézských souřadnicích. Tato skutečnost plyne z využití vizuálního řízení, jehož výstupem jsou právě zmíněné rychlosti. Okamžité rychlosti jsou matematicky reprezentovány pomocí tzv. twist vektoru, který je popsán ve tvaru *(5.1)*.

$$\boldsymbol{t} = \begin{bmatrix} \boldsymbol{v} \\ \boldsymbol{\omega} \end{bmatrix} \qquad (5.1)$$

Pro výpočet kloubových rychlostí z rychlosti TCP byl původně zvažován KDL::ChainIkSolverVel_pinv, dále jen PINV solver. Ten počítá kloubové rychlosti s využitím Moore-Penrose pseudoinverze Jacobiho matice $\boldsymbol{J}$. Jacobiho matice

reprezentuje, jak se změní póza TCP při změně natočení jednotlivých kloubů. Úpravou vztahu *(3.6)* získáme rovnici *(5.2)*.

$$\boldsymbol{v_{TCP}} = \boldsymbol{J}(\boldsymbol{q}) \cdot \dot{\boldsymbol{q}} \tag{5.2}$$

Rychlostní inverzní kinematika se zabývá opačným problémem. Hledá takové kloubové rychlosti $\dot{\boldsymbol{q}}$, které zajistí, že se TCP bude pohybovat předepsaným twistem v_{TCP}. Tento problém je možné matematicky formulovat vztahem *(5.3)*.

$$\dot{\boldsymbol{q}} = \boldsymbol{J}^{+}(\boldsymbol{q}) \cdot \boldsymbol{v_{TCP}} \tag{5.3}$$

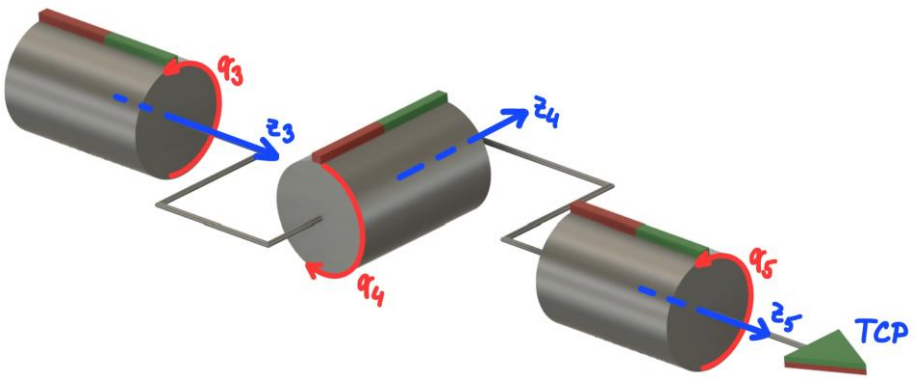
V tomto vztahu vystupuje matice $\boldsymbol{J}(\boldsymbol{q})^{+}$. Jedná se o Moore-Penrose pseudoinverzi Jacobiho matice. Použití klasické inverze Jacobiho matice $\boldsymbol{J}(\boldsymbol{q})^{-1}$ není vhodné z toho důvodu, že inverze obecně není odolná vůči singularitám v systému. Výpočet klasické inverze a je popsán vztahem *(5.4)*.

$$\boldsymbol{J^{-1}} = \frac{\mathrm{adj}(\boldsymbol{J})}{\det(\boldsymbol{J})} \tag{5.4}$$

V blízkosti singularit se $\det(\boldsymbol{J}) \to 0$ a $\boldsymbol{J}^{-1}$ není možné numericky spočítat vůbec, nebo by výsledek vyžadoval nekonečně vysoké rychlosti kloubů. Z toho důvodu místo klasické inverze $\boldsymbol{J}^{-1}$ použijeme pseudoinverzi $\boldsymbol{J}^{+}$. Ta vede na metodu nejmenších čtverců. V blízkosti singularit tak neselže, ale hledá takové řešení, jehož výsledkem jsou minimální kloubové rychlosti. Minimalizuje tedy normu $||\dot{\boldsymbol{q}}||$ při zachování požadovaného twistu. Vysvětlení samotné pseudoinverze překračuje rozsah této práce.

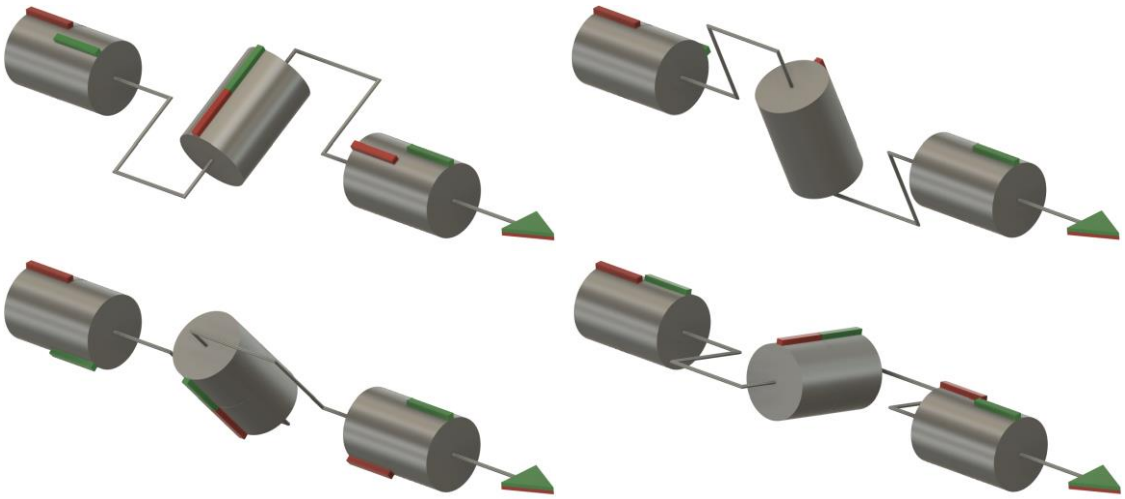
PINV solver byl původně nasazen hlavně vzhledem k jeho nízké výpočetní náročnosti. Při praktické implementaci se ale ukázalo, že solver není dostatečně robustní. V blízkosti singularit existuje více než jedno řešení inverzní kinematiky. Zmíněný solver, ale vždy poskytl právě takové řešení, které upřednostňovalo otáčení motoru totožným směrem. Při opakovaném průchodu přes singularitu se tak stalo, že se úhel natočení některého kloubu měnil vždy stejným směrem a kloub byl po sléze nucen zastavit na limitu svého rozsahu natočení.

Nejčastěji se vyskytující singularita v našem systému je singularita sférického zápěstí. Z Obr. 44 je zřejmé, že když bude úhel $q_4 = 0$, tak se osy z_3 a z_5 stanou kolineárními a splynou do jediné osy.



Obr. 44: Singularita sférického zápěstí

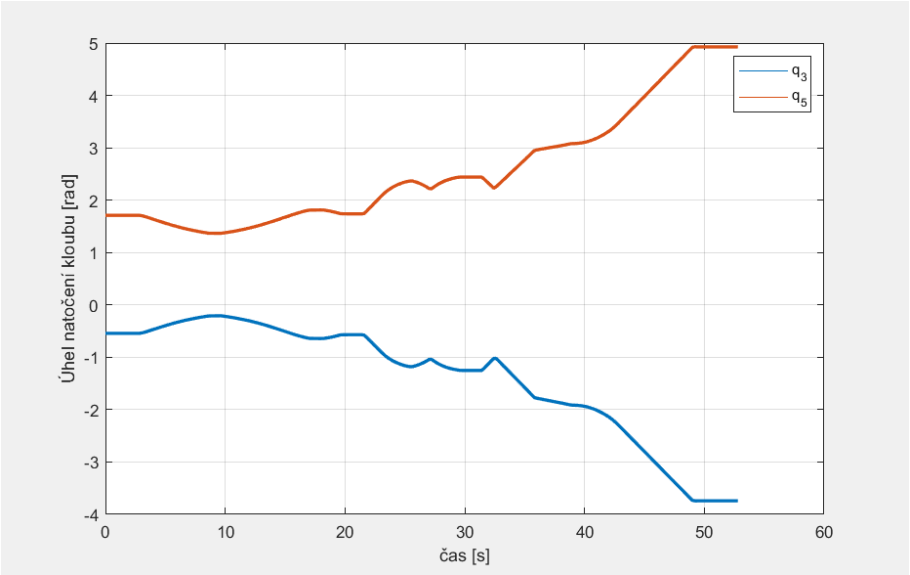
V tomto okamžiku má inverzní kinematika nekonečně mnoho řešení – existuje nekonečně mnoho různých kombinací natočení kloubů 3 a 5, které způsobí totožné natočení TCP, jak ukazuje Obr. 45.



Obr. 45: Řešení v singularitě

Stejná situace platí nejen pro úhly natočení kloubů, ale také pro jejich rychlosti. Například při požadavku na rotaci TCP kolem své osy z rychlostí $1\ \mathrm{rad/s}$ jsou validní řešení $(\dot{q}_3, \dot{q}_5) = (0, 1)\ \mathrm{rad/s}$, ale také $(\dot{q}_3, \dot{q}_5) = (-1, 2)\ \mathrm{rad/s}$, nebo v extrémním případě $(\dot{q}_3, \dot{q}_5) = (-1000, 1001)\ \mathrm{rad/s}$.

Tato skutečnost byla změřena a je zobrazena v Obr. 46. Můžeme zde pozorovat, že při průchodu přes singularitu PINV solver počítá takové řešení inverzní kinematiky, že se klouby q_3 a q_5 snaží působit proti sobě. Solver však většinou upřednostňuje rotaci q_3 v záporném směru a q_5 ve směru kladném. Zanedlouho tak oba klouby narazí na své limity rozsahu.



Obr. 46: Přechod přes singularitu s PINV solverem

Pro zlepšení chování v blízkosti singularit byl proto ve finální aplikaci nasazen solver KDL::ChainIkSolverVel_pinv_nso, dále jen NSO solver. Tento solver využívá tzv. nullspace.

Jacobiho matice obecně značí, jak se pohyby kloubů projeví na pohybu TCP. Když se v systému objeví singularita, tak se některé ze sloupců Jacobiho matice stanou vzájemně lineárně závislými a hodnost matice poklesne. Lineární závislost značí, že jeden pohyb TCP se dá vykonat různými klouby a řešení potom není jednoznačné, jak bylo popsáno výše. Platí věta o dimenzích jádra a obrazu (anglicky Rank-Nullity Theorem), která je popsána rovnicí (5.5). [42]

$$\dim(\boldsymbol{J}) = \mathrm{rank}(\boldsymbol{J}) + \mathrm{nullity}(\boldsymbol{J}) \tag{5.5}$$

Kde:

$\dim(\boldsymbol{J})$ – Dimenze Jacobiho matice, což odpovídá počtu kloubů

$\mathrm{rank}(\boldsymbol{J})$ – Hodnost matice $\boldsymbol{J}$, tedy počet lineárně nezávislých sloupců $\boldsymbol{J}$

$\mathrm{nullity}(\boldsymbol{J})$ – Rozměr jádra matice $\boldsymbol{J}$, tedy rozměr nullspace

Při poklesu hodnosti matice *J* vlivem singularity roste velikost nullspace. Nullspace je množina všech kloubových rychlostí, které sice mění vnitřní konfiguraci kinematického řetězce, ale nezpůsobí žádnou změnu polohy TCP. Zavedení nullspace nám dává možnost nasadit sekundární úlohu inverzní kinematiky, která je definována ve tvaru (5.6). [42] [43] [44]

$$\dot{\boldsymbol{q}}_2 = \alpha \cdot \nabla_q \boldsymbol{H}(\boldsymbol{q}) \tag{5.6}$$

Kde:

$\dot{\boldsymbol{q}}_2$ – Výstupní rychlosti sekundární úlohy

α – Síla sekundární úlohy

$\nabla_q \boldsymbol{H}(\boldsymbol{q})$ – Gradient výkonnostní funkce $\boldsymbol{H}(\boldsymbol{q})$

Podle tvaru funkce $\boldsymbol{H}(\boldsymbol{q})$ lze definovat několik typů sekundární úlohy. Za zmínku stojí například minimalizace energie kloubů nebo vyhýbání se překážkám. My však v naší aplikaci sekundární úlohu definujeme ve tvaru (5.7) Uvedený tvar sekundární úlohy počítá takové kloubové rychlosti, které se snaží klouby udržovat v blízkosti jejich optimálního natočení. [43] [44]

$$\boldsymbol{H}(\boldsymbol{q}) = -\frac{1}{2}\boldsymbol{W}\left(\boldsymbol{q} - \boldsymbol{q_{opt}}\right)^T\left(\boldsymbol{q} - \boldsymbol{q_{opt}}\right) \tag{5.7}$$

Kde:

$\boldsymbol{q}$ – Aktuální natočení kloubů

$\boldsymbol{q_{opt}}$ – Optimální natočení kloubů

$\boldsymbol{W}$ – Váhy jednotlivých kloubů

Po aplikování gradientu na funkci $\boldsymbol{H(q)}$, je rovnice (5.6) redukována na tvar (5.8).

$$\dot{\boldsymbol{q}}_{\boldsymbol{2}} = \alpha \boldsymbol{W}(\boldsymbol{q}_{\boldsymbol{opt}} - \boldsymbol{q}) \tag{5.8}$$

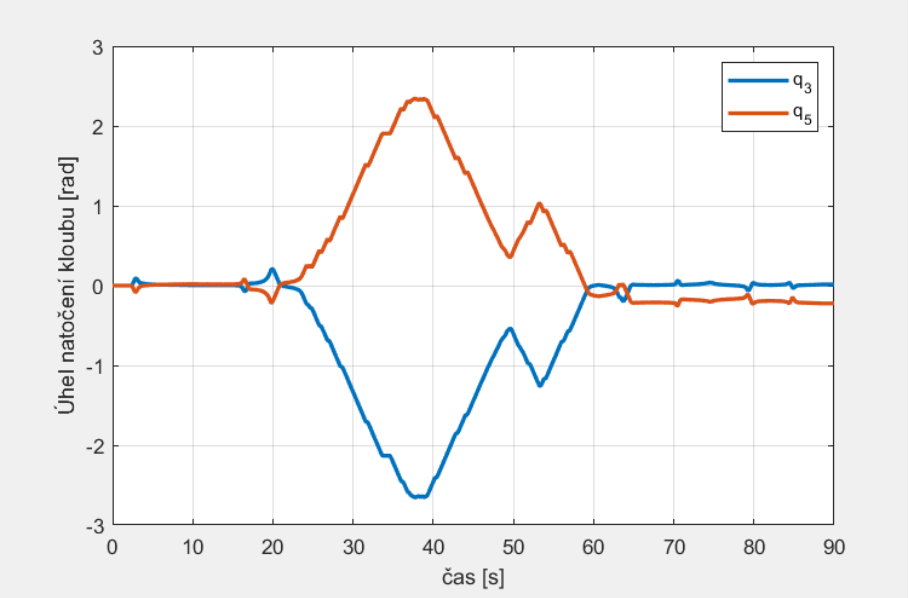
Primární úloha je tedy vypočtena klasickým způsobem a její výsledek je značen $\dot{\boldsymbol{q}}_{\boldsymbol{1}}$. Jediným rozdílem je, že NSO solver inverzi Jacobiho matice nepočítá pomocí pseudoinverze, ale tzv. SVD rozkladem. Vysvětlení tohoto algoritmu přesahuje rozsah práce. Výsledné kloubové rychlosti jsou potom složeny kombinací $\dot{\boldsymbol{q}}_{\boldsymbol{1}}$ a $\dot{\boldsymbol{q}}_{\boldsymbol{2}}$ ve tvaru (5.9).

$$\dot{\boldsymbol{q}} = \dot{\boldsymbol{q}}_{\boldsymbol{1}} + \dot{\boldsymbol{q}}_{\boldsymbol{2}} \tag{5.9}$$

Při aplikaci sekundární úlohy koncový bod manipulátoru stále vykonává zadané pohyby a není omezen. Výsledek sekundární úlohy totiž ovlivňuje pouze nullspace klouby. Tedy ty, jejichž pohyb nijak polohu TCP neovlivňuje, což jsou při singularitě zápěstí právě klouby 3 a 5. Zmíněný proces bývá označován jako nullspace optimalizace. To také značí písmena „nso“ v názvu solveru. [42] [43] [44]

NSO solver byl v rámci této práce nasazen a jeho efektivita byla změřena. Průchod přes singularitu je znázorněn v Obr. 47. Jako $\boldsymbol{q}_{\boldsymbol{opt}}$ je nastaven střed mezi horním a spodním limitem každého kloubu. Z průběhu je patrné, že přibližně v čase dvacet vteřin se začne projevovat singularita zápěstí a klouby tři a pět se začnou otáček opačnými směry. V okamžiku přibližně třicet osm vteřin vzdálenost obou kloubů od své optimální polohy vzroste na tolik, že se začne projevovat NSO a klouby jsou přivedeny zpět do optimálních poloh.

V grafu zároveň můžeme pozorovat, že se na konci pohybu 5. kloub nenachází přesně v nulové pozici. Jedná se o projev nespolehlivé konstrukce diferenciálního zápěstí. Této problematice se věnuje kapitola 1.3.4, přičemž názorný příklad je uveden na Obr. 12.



Obr. 47: Přechod přes singularitu s NSO solverem

NSO solver provádí v každém kroku znatelně více operací než solver PINV. Jeho výpočetní náročnost by tedy měla být podstatně vyšší. V průběhu měření však doba výpočtu solverem PINV odpovídala průměrně 85.1 μs, zatímco solver NSO dosahoval průměrné doby 98.5 μs. Tento rozdíl je tedy zanedbatelný.

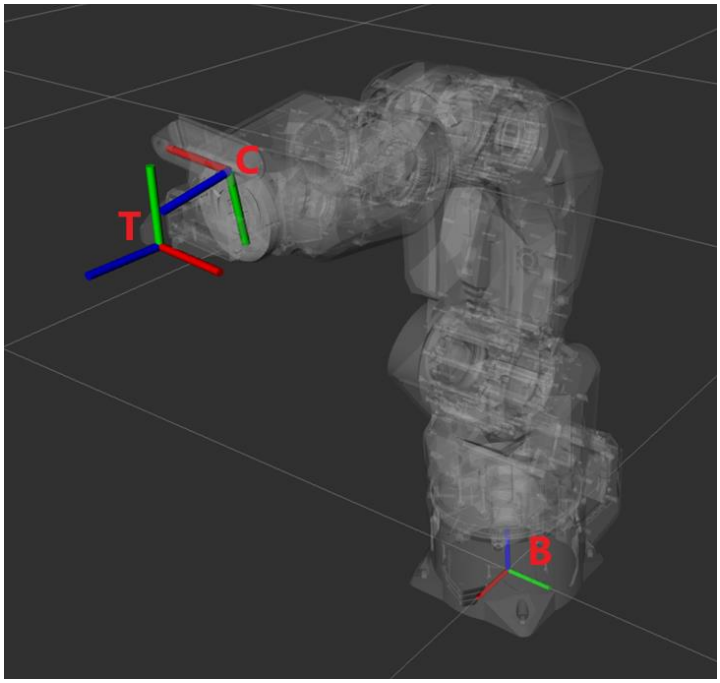
Využitý solver nebere v úvahu limity jednotlivých kloubů a kolize je tak nutné kontrolovat ručně. Stejně tak solver nekontroluje omezení na maximální rychlosti kloubů. Tato skutečnost představuje poměrně velký problém hlavně z toho důvodu, že právě omezení rychlostí jsou vlivem slabé konstrukce výrazná. Bez kontroly maximální rychlosti docházelo k tomu, že překročení rychlosti nějakého kloubu kontroloval až driver, který ji poté omezil. Tím ale došlo k zásahu do kinematiky a TCP se poté pohyboval nekorektně. Řešením je detekovat kloub, který překročil rychlost. Poté vypočíst poměr, o který hodnota limit přesáhla a tímto poměrem vynásobit všechny výstupní kloubové rychlosti. TCP se tak v těchto okamžicích sice nebude pohybovat zadanou rychlostí, ale svoji trajektorii si zachová. A to je pro naši aplikaci důležitější.

Vizuální řízení poskytuje jako akční zásah twist kamery v_{cam}. Rychlostní inverzní kinematika ale v základu dokáže řídit pouze rychlost koncového bodu manipulátoru $\boldsymbol{v_{TCP}}$. Mezi umístěním kamery a TCP platí transformace $\boldsymbol{p_{TCP} = H_{TC} \cdot p_{cam}}$. Transformace $\boldsymbol{H_{TC}}$ vyjadřuje pevnou vazbu mezi senzorem kamery a TCP. Jelikož je vazba pevná, pro transformaci rychlostí z $\boldsymbol{v_{cam}}$ na $\boldsymbol{v_{TCP}}$ není možné přímo použít Jacobiho matici, která popisuje závislost na kloubových rychlostech – tato vazba žádné klouby neobsahuje. Místo toho je nutné využít principy Lieovy algebry a aplikovat tzv. adjungovanou transformaci $\mathrm{Ad}_{\boldsymbol{H}}$ (anglicky Adjoint representation of a transformation). Pozor, nejedná se o adjungovanou maticí $\mathrm{adj}(\boldsymbol{H})$. Při definici twistu popsané v rovnici (5.1) má adjungovaná transformace $\mathrm{Ad}_{\boldsymbol{H}}$ tvar (5.10) a platí rovnice (5.11). Tento vztah popisuje transformaci twistu kamery $\boldsymbol{v_{cam}}$ na twist TCP $\boldsymbol{v_{TCP}}$ za předpokladu, že definována transformace $\boldsymbol{H_{TC}}$ má tvar popsaný v *(3.2)*. [7] [45]

$$\mathrm{Ad}_{\boldsymbol{H}} = \begin{bmatrix} \boldsymbol{R} & \boldsymbol{T} \times \boldsymbol{R} \\ \boldsymbol{0} & \boldsymbol{R} \end{bmatrix} \tag{5.10}$$

$$\boldsymbol{v_{tcp}} = \begin{bmatrix} \boldsymbol{v_{TCP}} \\ \boldsymbol{\omega_{TCP}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{R} & \boldsymbol{T} \times \boldsymbol{R} \\ \boldsymbol{0} & \boldsymbol{R} \end{bmatrix} \begin{bmatrix} \boldsymbol{v_{cam}} \\ \boldsymbol{\omega_{cam}} \end{bmatrix} = \mathrm{Ad}_{\boldsymbol{H_{TC}}} \cdot \boldsymbol{v_{cam}} \tag{5.11}$$

Twist kamery, který poskytuje vizuální řízení je navíc v lokálním souřadnicovém systému kamery. Ten je v Obr. 48 označen písmenem C. Rychlostní solver však dokáže přijímat požadavky pouze z globálního systému báze – B.



Obr. 48: Souřadnicové systémy manipulátoru

Pro přepočet rychlostí kamery z lokálního systému C do globálního systému B je nutné provést operaci popsanou rovnicí (5.12). [46]

$$^{B}\boldsymbol{v}_{cam} = {}^{B}\boldsymbol{R}_{C} \cdot {}^{C}\boldsymbol{v}_{cam} \tag{5.12}$$

Kde:

$^{C}\boldsymbol{v}_{cam}$ – Twist kamery v lokálním souř. systému C – výstup PBVS

$^{B}\boldsymbol{v}_{cam}$ – Twist kamery v globálním souř. systému B – pro KDL solver

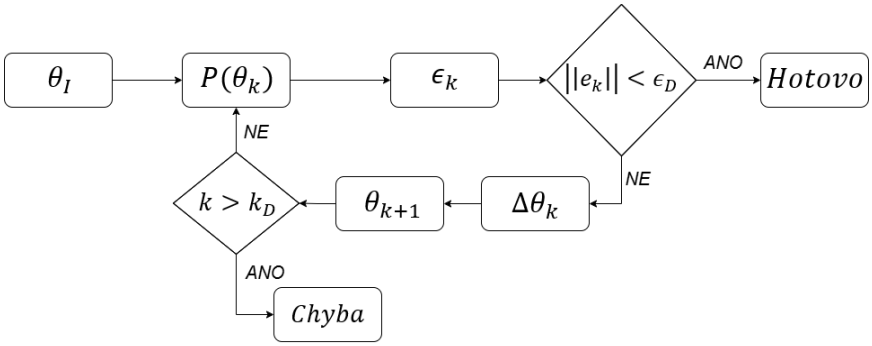
$^{B}\boldsymbol{R}_{C}$ – Aktuální matice rotace mezi lokálním a globálním souř. systémem

O tom, jestli požadavek na twist patří pro TCP nebo kameru a jestli je v globálním nebo lokálním souřadnicovém systému rozhoduje kolonka „frame_id“ v ROS2 zprávě pro uzel kinematiky. Mezi platné možnosti patří camera, base-camera, tool, base-tool. Pokud frame_id není vyplněn, je použita základní varianta base-tool.

5.3.3 Inverzní kinematika – póza TCP

Inverzní kinematika pózy se v rámci této aplikace aktivně neuplatňuje, přesto je implementována, aby řešení kinematiky bylo kompletní a využitelné i pro jiné nasazení manipulátoru než samostatné vizuální řízení. Přijímá požadovanou pózu TCP a vrací nutné úhly natočení jednotlivých kloubů.

Implementace využívá solver KDL::ChainIkSolverPos_NR_JL. Písmena NR zde značí metodu výpočtu. Tou je metoda Newton-Raphson, též označována jako Newtonova iterační metoda. Iteračně upravuje odhad kloubových úhlů s využitím inverze Jacobiho matice. Algoritmus je popsán v Obr. 49.



Obr. 49: Newton-Raphson algoritmus

V prvním kroku předán počáteční odhad kloubových úhlů. Ten je značen jako $\boldsymbol{\theta_I}$. Na základě těchto úhlů je pomocí přímé kinematiky vypočtena odhadovaná póza TCP $\boldsymbol{P(\theta_k)}$. Ta je porovnána s požadovanou pózou TCP $\boldsymbol{P_D}$ a je vypočtena odchylka $\boldsymbol{\epsilon_k}$ podle rovnice *(5.13)*. Pokud je norma této odchylky menší, než je požadovaná přesnost ϵ_D, algoritmus se ukončí a předává úhly $\boldsymbol{\theta_k}$ jako řešení. V opačném případě je vypočtena změna úhlů $\boldsymbol{\Delta\theta_k}$ pomocí inverze Jacobiho matice $\boldsymbol{J(\theta_k)^{-1}}$, jak je zobrazeno v rovnici *(5.14)*. Tato změna kloubových úhlů je dosazena do rovnice *(5.15)* a nové kloubové úhly $\boldsymbol{\theta_{k+1}}$ jsou v další iteraci dosazeny do úlohy přímé kinematiky $\boldsymbol{P(\theta_k = \theta_{k+1})}$. Pokud počet iterací přesáhne povolenou hranici, algoritmus se ukončí a vyvolá chybovou hlášku.

Použitý solver ve skutečnosti nepočítá klasickou inverzi $\boldsymbol{J^{-1}}$, ale problém předává solveru rychlostní inverzní kinematiky, který počítá Moore-Penrose pseudoinverzi Jacobiho matice $\boldsymbol{J^{+}}$. [44]

$$\boldsymbol{\epsilon_k = P_D - P(\theta_k)} \qquad \textit{(5.13)}$$

$$\boldsymbol{\Delta\theta_k = J(\theta_k)^{-1} \cdot \epsilon_k} \qquad \textit{(5.14)}$$

$$\boldsymbol{\theta_{k+1} = \theta_k + \Delta\theta_k} \qquad \textit{(5.15)}$$

Solver KDL::ChainIkSolverPos_NR_JL bere v úvahu kloubové limity. To značí písmena JL (Joint Limits) v názvu. Pokud během výpočtů solver zjistí, že vypočtené $\boldsymbol{\theta_{k+1}}$ je mimo limity daného kloubu, omezí danou hodnotu na hodnotu limitu a pokračuje ve výpočtech. Solver bohužel nedokáže pracovat s více hypotézami současně. Nedokáže najít alternativní konfigurace, pokud vlivem limitů nenajde řešení. Jednoduše se snaží dopočítat rozpracované řešení a když narazí na limit, tak hodnotu omezí a pokračuje dál. [44]

Přestože nasazený solver hlídá limity kloubů, stávalo se, že v rámci limitů vypočetl celé otáčky kloubů. Proto bylo potřeba provést korekci, která redukuje celé otáčky kloubu. Korekce je popsána rovnicí (5.16).

$$\boldsymbol{\theta} = \boldsymbol{\theta} \bmod (2\pi) \qquad (5.16)$$

Výsledkem KDL solveru jsou nové úhly natočení kloubů. Byla doplněna funkce calc_uniform_speed, která na základě aktuálních a nově vypočtených kloubových úhlů

dopočítá také rychlosti všech kloubů. Cílem je zajistit plynulé a rovnoměrné pohyby. Díky tomu dorazí všechny klouby do své cílové pozice ve stejný okamžik.

5.4 Kamera

O sběr dat z kamery se stará uzel CameraDriver. Jeho komunikační interface je popsán Tab. 6. Uzel využívá funkce jak platformy ViSP, tak knihovny librealsense. Librealsense je oficiální knihovna vydaná společností Intel pro přímou komunikaci s RealSense kamerami. Díky přímému přístupu ke všem funkcím kamery má uživatel kontrolu nad způsobem zpracování dat. Uzel poskytuje také informace o intrinsických parametrech kamery, které jsou nezbytné pro kalibraci.

Tab. 6: Komunikační rozhraní uzlu CameraDriver

Role	Topic	Komunikační partner	Datový typ	Popis
Publisher	camera/rgb	GUI, ObjectDetection	Image	Barevné snímky
Publisher	camera/depth	ObjectDetection (zatím nevyužito)	Image	Hloubkové snímky
Service server	camera/params	ObjectDetection	CameraParams*	Intrinsické parametry kamery

Platforma ViSP poskytuje nadstavbu nad základní librealsense. Tou je třída vpRealsense2, která integruje kameru přímo do platformy ViSP a následné zpracování v rámci systému je tak jednodušší. Třída vpRealSense2 však neumožnuje přímou správu datového toku. Pokud chceme vyčítat data plynule, musíme využít třídu rs::pipeline ze základní knihovny librealsense a snímky konvertovat do ViSP formátu pomocí vpImageConvert.

Pro přenos snímků jsou využity zprávy typu Image z knihovny sensor_msgs. Parametry kamery poskytuje ostatním uzlům service server s vlastním interface CameraParams. Použitá kamera RealSense D435 při žádosti o kalibrační data poskytuje pouze model a parametry zkreslení a matici vnitřních parametrů $\boldsymbol{K}$. Tato matice je popsána v kapitole 5.9.5.

5.5 Detekce pózy objektu

V této aplikaci jsou sledované objekty označeny značkami typu AprilTag. Jedná se o typ binární vizuální značky navržený pro rychlou a robustní detekci kamerovými systémy. Díky zmíněným vlastnostem je v knihovně ViSP implementována nativní podpora detekce AprilTagů. Funkce vpDetectorAprilTag::detect na základě intrinsických parametrů kamery a definované velikosti značky detekuje z předaného snímku 3D pózu značky v rámci souřadnicového systému kamery. Pokud je ve snímku několik značek, systém se zaměří na značku nejbližší.

Komunikaci v rámci ROS2 Network zajišťuje uzel ObjectDetection. Jeho komunikační rozhraní je zobrazeno v Tab. 7. Uzel přijímá 2D snímky, ve kterých detekuje 3D pózu značky. Tu následně publikuje. Pro spolehlivou detekci je nutná přesná znalost intrinsických parametrů kamery. V rámci prvního cyklu jsou tak tyto parametry z uzlu CameraDriver vyžádány pomocí service klienta.

Tab. 7: Komunikační rozhraní uzlu ObjectDetection

Role	Topic	Komunikační partner	Datový typ	Popis
Subscriber	camera/rgb	CameraDriver	Image	Barevné snímky
Publisher	detector/ object_pose	GUI, VisualServoing	Image	Hloubkové snímky
Service server	camera/params	CameraDriver	CameraParams*	Intrinsické parametry kamery

5.6 Hand-eye kalibrace

Uzel HandEyeCalib zajišťuje hand-eye kalibraci kamery připevněné k manipulátoru. Z uzlu ObjectDetection přijímá aktuální pózu detekované značky v kameře. Dalším datovým vstupem je aktuální póza TCP, kterou získává z uzlu Kinematics. Uzel také HandEyeCalib provozuje service server, který čeká na povel pro uložení zmiňovaných dat do YAML souborů. Komunikační rozhraní uzlu je popsáno v Tab. 8

Tab. 8: Komunikační rozhraní uzlu HandEyeCalib

Role	Topic	Komunikační partner	Datový typ	Popis
Subscriber	kinematics/state_pose	Kinematics	PoseStamped	Póza TCP
Subscriber	detector/object_pose	ObjectDetection	PoseStamped	Póza objektu v kameře
Service server	calib/execute_h_e	CLI	Trigger	Vyvolává uložení dat

5.7 Vizuální řízení

Vizuální řízení zajišťuje uzel VisualServoing, který je postavený na platformě ViSP. Jako vstupní datové toky přijímá dvě matice homogenní transformace cMo a $cdMo$. ViSP značí aktuální pózu kamery jako c a požadovanou pózu kamery jako cd (camera desired). Matice homogenní transformace ze systému x do systému y jsou poté označovány písmeny yMx. Stejné značení bude respektováno v tomto textu. Matice $\boldsymbol{cMo}$ tedy reprezentuje **aktuální pózu** objektu v kameře a matice $\boldsymbol{cdMo}$ značí **požadovanou pózu** objektu v kameře. Cílem úlohy je vypočítat rychlost kamery v_c, která zajistí minimalizaci odchylky mezi $cdMo$ a cMo. Kompletní komunikační rozhraní uzlu VisualServoing je popsáno v Tab. 9

Tab. 9: Komunikační rozhraní uzlu VisualServoing

Role	Topic	Komunikační partner	Datový typ	Popis
Subscriber	detector/object_pose	ObjectDetection	PoseStamped	Póza objektu v kameře cMo
Subscriber	visual_servoing/ req_object_pose	Gui	PoseStamped	Požadovaná póza objektu v kameře $cdMo$
Publisher	kinematics/set_velocity	Kinematics	TwistStamped	Rychlost kamery v_c
Action server	path_planning/ execute_path	VisualServoing	ExecutePath*	Vykonání naplánované trajektorie

5.7.1 Inicializace

Během inicializace je nutné plně definovat úlohu řízení, kterou v platformě ViSP zajišťuje třída vp::task.

Vizuální řízení definuje regulační odchylku jako $\boldsymbol{e}(t) = \boldsymbol{s}(\boldsymbol{m}(t), \boldsymbol{a}) - \boldsymbol{s}^*$, jak zmiňuje rovnice (4.1). V kapitole 4.1 je popsáno, že metoda PBVS definuje vizuální charakteristiky $\boldsymbol{s}$ a $\boldsymbol{s}^*$ jako sadu 3D parametrů, které jsou odvozeny z obrazu. Vizuální charakteristiky tedy představují vektor popsaný rovnicí (5.17).

$$\boldsymbol{s} = \begin{pmatrix} \boldsymbol{s_t} \\ \boldsymbol{s_{\theta u}} \end{pmatrix} \tag{5.17}$$

Kde:

$\boldsymbol{s_t}$ – Vektor translace z c do cd

$\boldsymbol{s_{\theta u}}$ – Rotace z c do cd reprezentována ve tvaru úhel-osa

Čtyřprvková reprezentace rotace ve tvaru $\boldsymbol{\theta u}$, též označováno jako rotace úhel-osa je popsána rovnicí (5.18).

$$\boldsymbol{\theta u} = \left(u_x, u_y, u_z, \theta\right)^T \tag{5.18}$$

Kde:

$\left(u_x, u_y, u_z\right)^T$ – Je jednotkový vektor udávající osu rotace

θ – Míra rotace

Když je vektor u jednotkový, můžeme jednotlivé prvky vektoru roznásobit úhlem θ a tvar se redukuje na tři čísla jako je parné z rovnice (5.19).

$$\boldsymbol{\theta u} = (\theta_x, \theta_y, \theta_z) \tag{5.19}$$

Tato reprezentace 3D rotace na rozdíl od reprezentace Eulerovými úhly nepodléhá tzv. gimball locku. Ve srovnání s quaterniony je reprezentována pomocí tří reálných čísel místo čtyř. To samé platí při srovnání s maticí rotace, která obsahuje reálných čísel devět.

Přepočet mezi maticí rotace $\boldsymbol{R}$ a reprezentací úhel-osa popisuje Rodriguesova rovnice (5.20) [45].

$$\boldsymbol{R} = e^{[\boldsymbol{u}]\boldsymbol{\theta}} = \boldsymbol{I} + \sin(\theta)\,[\boldsymbol{u}] + (1 - \cos(\theta))[\boldsymbol{u}]^2 \tag{5.20}$$

V rámci vizuálního řízení definujeme tzv. interakční matici $\boldsymbol{L_e}$. Často bývá označována také jako citlivostní matice vizuálních rysů, či obrazová Jacobiho matice. Tato matice vyjadřuje míru změny vizuálních rysů při změně pózy kamery. Platí tedy rovnice (5.21) a (5.22). [42]

$$\dot{\boldsymbol{s}} = \boldsymbol{L_e}(\boldsymbol{p_{cam}}) \cdot \dot{\boldsymbol{p}}_{\boldsymbol{cam}} \tag{5.21}$$

$$\boldsymbol{L_e}(\boldsymbol{p_{cam}}) = \frac{\partial \boldsymbol{s}(\boldsymbol{p_{cam}})}{\partial \boldsymbol{p_{cam}}} \tag{5.22}$$

Kde:

$\boldsymbol{L_e}$ – Interakční matice
$\dot{\boldsymbol{p}}_{\boldsymbol{cam}}$ – Změna pózy kamery
$\dot{\boldsymbol{s}}$ – Změna vizuálních rysů

Při umístění kamery v konfiguraci eye-in-hand je řídicí zákon definován rovnicí (5.23). [18] [47] [42]

$$\boldsymbol{v_c} = -\lambda \boldsymbol{L_e^+ e} \tag{5.23}$$

Kde:

$\boldsymbol{v_c}$ – Výsledná rychlost kamery
λ – Zisk regulátoru
$\boldsymbol{L_e^+}$ – Pseudoinverze interakční matice L_e

Zisk regulátoru může být buď pevně nastaven na požadovanou konstantu, nebo se může adaptivně měnit. Adaptivní korekce λ zajišťuje metoda vpAdaptiveGain, která v průběhu vykonávání úlohy řízení koriguje hodnotu zisku podle rovnice (5.24). [47]

$$\boldsymbol{\lambda}\big(\big|\big|\boldsymbol{e}\big|\big|\big) = (\lambda_0 - \lambda_\infty) \cdot \exp\left(-\frac{\lambda_0'}{\lambda_0 - \lambda_\infty}\big|\big|\boldsymbol{e}\big|\big|\right) + \lambda_\infty \tag{5.24}$$

Kde:

$\big|\big|\boldsymbol{e}\big|\big|$ – Norma regulační odchylky – v této aplikaci využita norma L^∞
λ_0 – Zisk regulátoru pro $\big|\big|\boldsymbol{e}\big|\big| \to 0$
λ_∞ – Zisk regulátoru pro $\big|\big|\boldsymbol{e}\big|\big| \to \infty$
λ_0' – Směrnice regulátoru pro $\big|\big|\boldsymbol{e}\big|\big| \to 0$

Vhodným nastavením zmíněných parametrů adaptivního zisku můžeme upravovat plynulost, preciznost a rychlost regulace. Dokumentace uvádí běžné hodnoty λ v rozsahu

desítek. V rámci této práce jsou však parametry nastaveny na hodnoty $\lambda_0 = 0.3$, $\lambda_\infty = 0.1$ a $\lambda_0' = 10$. Tyto konstanty byly určeny empiricky a vychází z omezení konstrukce manipulátoru. Při vyšších hodnotách λ se manipulátor vlivem volnosti konstrukce při jakémkoliv pohybu rozkmital a k jeho stabilizaci nikdy nedošlo. Naopak nižší hodnoty λ vedly k tak nízkým akčním zásahům, že se motory nultého kloubu a diferenčního zápěstí nedokázaly roztočit vlivem nízkého převodového poměru.

Po nastavení veškerých zmíněných parametrů je provedena jejich validace metodou task.testInitialization.

5.7.2 Průběh řízení

Po úspěšné inicializaci je cyklicky vykonávána definovaná úloha řízení. Nekonečná smyčka řízení běží na odděleném vlákně, aby neblokovala komunikaci v rámci ROS2 Network. V každém cyklu je nutné spočítat matici $\boldsymbol{cdMc}$, která značí transformaci z aktuální pózy kamery do požadované pózy kamery. Platí rovnice (5.25).

$$\boldsymbol{cdMc} = \boldsymbol{cdMo} \cdot \boldsymbol{cMo}^{-1} \tag{5.25}$$

Z matice $\boldsymbol{cdMc}$ jsou odvozeny nové vizuální charakteristiky $\boldsymbol{s_t}$ a $\boldsymbol{s_{\theta u}}$. Pro připomenutí, matice $\boldsymbol{cdMc}$ je matice homogenní transformace a má tvar *(3.2)* Skládá se tedy, mimo jiné, z matice rotace $cdRc$ a translačního vektoru $\boldsymbol{cdTc}$. Pro určení $\boldsymbol{s_{\theta u}}$ a $\boldsymbol{s_t}$ potom platí vztahy (5.26) až (5.33). [45]

$$\mathrm{tr}(\boldsymbol{R}) = 1 + 2\cos(\theta) \tag{5.26}$$

$$\theta = -\cos^{-1}\left(\frac{\mathrm{tr}(\boldsymbol{cdRc}) + 1}{2}\right) \tag{5.27}$$

$$[\boldsymbol{u}] = \begin{pmatrix} 0 & -u_z & u_y \\ u_z & 0 & -u_x \\ -u_y & u_x & 0 \end{pmatrix} = \frac{1}{2\sin(\theta)}(\boldsymbol{cdRc} - \boldsymbol{cdRc}^T) \tag{5.28}$$

$$u_x = \frac{1}{2\sin(\theta)}(r_{32} - r_{23}) \tag{5.29}$$

$$u_y = \frac{1}{2\sin(\theta)}(r_{13} - r_{31}) \tag{5.30}$$

$$u_z = \frac{1}{2\sin(\theta)}(r_{21} - r_{12}) \tag{5.31}$$

$$\boldsymbol{s_{\theta u}} = \left(\theta u_x, \theta u_y, \theta u_z\right)^T \tag{5.32}$$

$$\boldsymbol{s_t} = \boldsymbol{cdTc} \tag{5.33}$$

Poté je vypočtena regulační odchylka e, nová interakční matice $\boldsymbol{L_e}$ (5.22), adaptivní zisk λ (5.24) a výsledná rychlost kamery $\boldsymbol{v_{cam}}$ (5.23). Tato rychlost kamery je následně publikována na topicu kinematics/set_velocity a celý postup se opakuje.

Pokud je norma odchylky $\boldsymbol{e}$ menší než požadovaná odchylka e_d, nastaví se příznak on_pose, který se uplatňuje v procesu vykonávání trajektorie. Jedná se o sdílenou atomickou proměnnou typu bool. Aby se předešlo race conditions, je přístup k této proměnné zajištěn synchronizačním mechanismem std::lock_guard. Ten zajišťuje automatické zamykání a odemykání objektu std::mutex. Mutex (mutual exclusion – vzájemné vyloučení) slouží k zajištění exkluzivního přístupu ke sdíleným zdrojům.

5.7.3 Vykonání trajektorie

O vykonávání předané trajektorie se stará action server, který běží v uzlu VisualServoing. Jako požadavek dostává trajektorii ve formě pole póz PoseStamped. Poté postupně přepisuje matici $\boldsymbol{cdMo}$, která se využívá v algoritmu PBVS. Matice $\boldsymbol{cdMo}$ představuje zdroj, který je sdílený mezi více vlákny, proto je zápis nutné ošetřit mechanismem std::lock_guard, který zamyká std::mutex dané proměnné.

K přepisu matice $\boldsymbol{cdMo}$ na další krok dochází vždy, když smyčka vykonávající PBVS nastaví on_pose na hodnotu true. Ve stejném okamžiku je odeslána průběžná zpětná vazba action serveru (feedback). Poté, co jsou dosaženy všechny pózy z pole trajektorie, odesílá action server informaci o úspěšném vykonání trajektorie. V případě, že je kdykoliv v průběhu vykonávání daná akce přerušena, action server odesílá informaci o tomto přerušení.

5.8 Plánovač trajektorie

Plánování trajektorie zajišťuje balík PathPlanning. Jeho komunikační rozhraní v rámci ROS2 Network popisuje Tab. 10.

Tab. 10: Komunikační rozhraní uzlu PathPlanning

Role	Topic	Komunikační partner	Datový typ	Popis
Service server	path_planning/ lerp	GUI	GetPlan	Vytvoření offline trajektorie mezi počáteční a koncovou pózou

Teorie spojená s problematikou plánování trajektorie je popsána v kapitole 4.5. Aktuální kapitola se zabývá finální implementací. Je nasazen jednoduchý offline plánovač, který vytváří lineární trajektorie s využitím algoritmů LERP a SLERP. Zkratka LERP popisuje linear interpolation (lineární interpolace). Slouží k vytvoření přímky ve 3D prostoru mezi dvěma body a je popsána vzorcem (5.34). Výsledkem je skupina bodů s rovnoměrným krokem k.

$$\boldsymbol{p}(k) = (1 - k) \cdot \boldsymbol{p_0} + \mathrm{k} \cdot \boldsymbol{p_1} \tag{5.34}$$

Kde:

$\boldsymbol{p_0}$ – Vektor počáteční polohy

$\boldsymbol{p_1}$ – Vektor cílové polohy

SLERP potom popisuje název spherical linear interpolation (v překladu sférická lineární interpolace). SLERP slouží pro výpočet nejkratší cesty mezi počáteční a koncovou orientací, které jsou vyjádřeny kvaterniony. Pohyb po vygenerované trajektorii je možné popsat jako rotaci kolem pevné osy s rovnoměrnou rotační rychlostí. [48] Výpočet SLERP ukazuje rovnice (5.35)

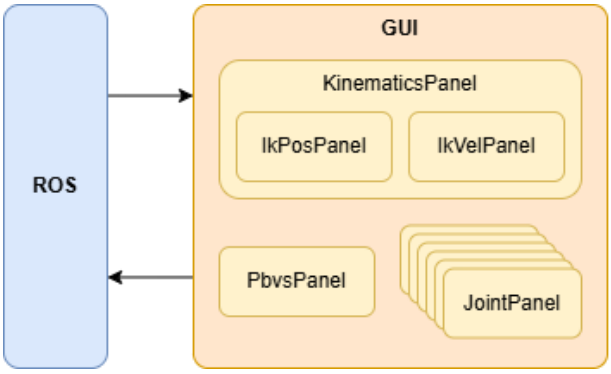
$$\boldsymbol{q}(k) = \frac{\sin\big((1-k)\cdot\theta\big)}{\sin(\theta)} \cdot \mathbf{q_0} + \frac{\sin(k\cdot\theta)}{\sin(\theta)} \cdot \boldsymbol{q_1} \tag{5.35}$$

Kde:

- $\boldsymbol{q_0}$ – Počáteční orientace vyjádřená kvaternionem
- $\boldsymbol{q_1}$ – Cílová orientace vyjádřená kvaternionem
- θ – Úhel rotace mezi $\boldsymbol{q_0}$ a $\boldsymbol{q_1}$. Platí $\cos(\theta) = \boldsymbol{q_0} \cdot \boldsymbol{q_1}$

5.9 Grafické uživatelské rozhraní

Podstatnou část této práce představuje také grafické uživatelské rozhraní (anglicky Graphical User Interface – dále jen GUI) realizované uzlem GuiNode. GUI je postavené na knihovně tkinter. Jedná se o standardní knihovnu jazyka Python právě pro tvorbu grafických rozhraní. Architektura kódu je rozdělena na dvě hlavní sekce – grafická a ROS. ROS2 sekce slouží pro komunikaci s ostatními uzly v rámci ROS2 Network. Grafická část poté vizualizuje přijímaná data a umožňuje uživateli zadávat příkazy. Zjednodušené schéma architektury tohoto uzlu je zobrazeno v Obr. 50. Komunikační rozhraní v rámci ROS2 Network potom ukazuje Tab. 11.



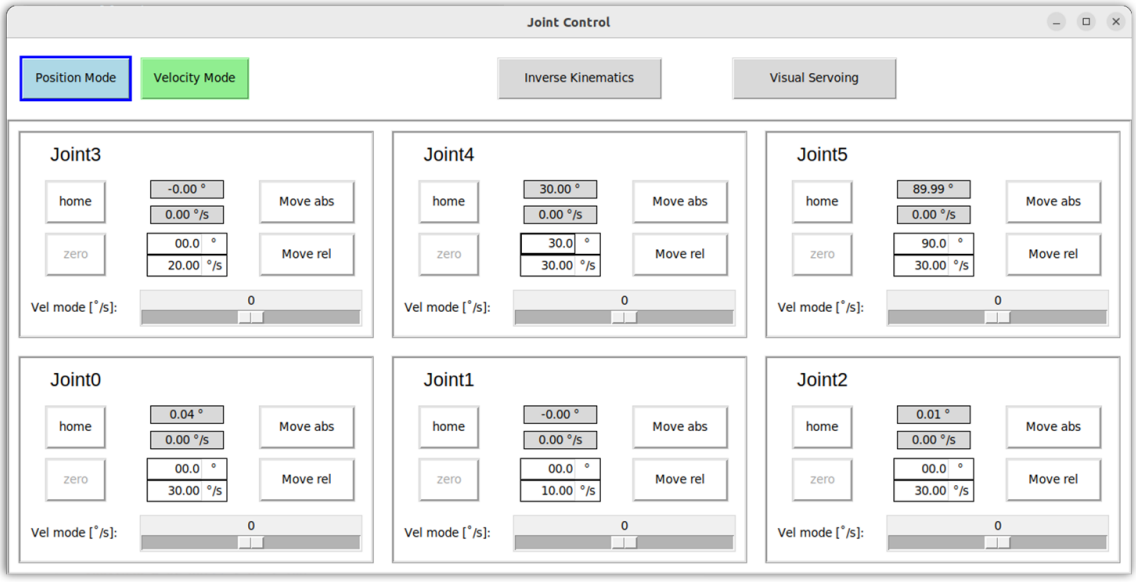
Obr. 50: Schéma architektury GuiNode

Tab. 11: Komunikační rozhraní uzlu GuiNode

Role	Topic	Komunikační partner	Datový typ	Popis
Subscriber	manipulator/ state_joints_position	ManipulatorDriver	JointState	Aktuální natočení kloubů
Subscriber	kinematics/ state_pose	Kinematics	PoseStamped	Aktuální póza TCP
Subscriber	detector/ object_pose	ObjectDetection	PoseStamped	Aktuální póza objektu v kameře
Subscriber	camera/ rgb	CameraDriver	Image	Snímky z kamery
Publisher	manipulator/ set_joints_position	ManipulatorDriver	JointState	Požadované natočení kloubů
Publisher	manipulator/ set_joints_velocity	ManipulatorDriver	JointState	Požadované rychlosti kloubů
Publisher	kinematics/ set_pose	Kinematics	PoseStamped	Požadovaná póza TCP
Publisher	kinematics/ set_velocity	Kinematics	TwistStamped	Požadovaná rychlost TCP/kamery
Publisher	visual_servoing/ req_object_pose	VisualServoing	PoseStamped	Požadovaná póza objektu v kameře
Service client	manipulator/ home_single_joint	ManipulatorDriver	HomeAxis*	Inicializace kloubu
Service client	manipulator/ zero_single_joint	ManipulatorDriver	ResetAxis*	Nulování kloubu
Service client	manipulator/ change_mode	ManipulatorDriver	ChangeMode*	Přepnutí režimu řízení manipulátoru
Service client	path_planning/ lerp	PathPlanning	GetPlan	Požadavek o vytvoření trajektorie
Action client	path_planning/ execute_path	VisualServoing	ExecutePath*	Požadavek o provedení trajektorie

5.9.1 Hlavní obrazovka

Hlavní obrazovka GUI je znázorněna v Obr. 51. Skládá se ze šesti panelů pro přímou interakci s jednotlivými klouby manipulátoru. Dále obsahuje tlačítka pro volbu režimu řízení a dvě tlačítka pro otevření vyskakovacích (pop-up) oken pro ovládání inverzní kinematiky a vizuálního řízení.



Obr. 51: GUI – Hlavní obrazovka

Pro změnu režimu řízení všech kloubů slouží tlačítka *Position Mode* a *Velocity Mode*. Podle aktuálního režimu jsou následně aktivována a deaktivována ostatní tlačítka GUI. Aktuální režim řízení je znárodněn tučným rámečkem, jak ukazuje Obr. 52.

Obr. 52: GUI – Přepínání režimů

5.9.2 Přímé řízení kloubů

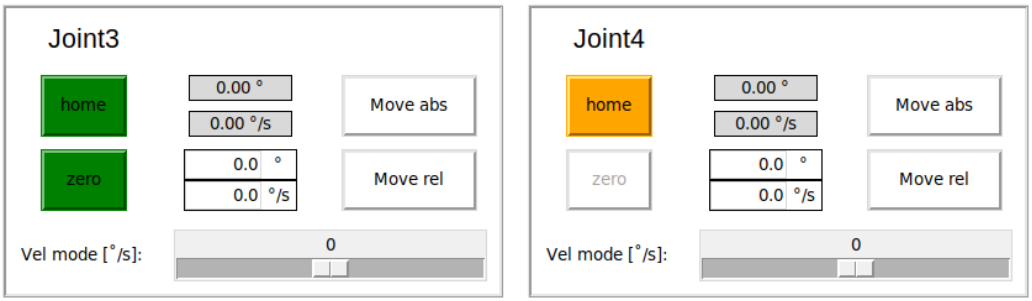
Při návrhu grafického rozhraní byly silně využívány vlastní šablony. U panelů nebyla programována jednotlivá tlačítka, ale vznikla jedna šablona formou vlastní třídy. Panel pro řízení kloubu je detailně zobrazen v Obr. 53. Jednotlivé panely poté reprezentují instance této šablony (třídy) umístěné do mřížky.

Obr. 53: GUI – Panel kloubu. Poziční režim vlevo, rychlostní režim vpravo

Panel kloubu umožňuje inicializaci a nulování kloubu. Dále zobrazuje aktuální polohu a rychlost. Uživatel zadává požadovanou polohu a rychlost, kterou má kloub dosáhnout během pohybu při pozičním řízení. Tlačítka *Move abs* a *Move rel* slouží pro vyvolání požadavku na absolutní pohyb vzhledem k nule, nebo relativní pohyb vzhledem

k aktuální poloze. Když je aktivní poziční režim řízení, jsou povolena všechna tlačítka, kromě posuvníku ve spodní části. Tento posuvník je aktivní v rychlostním režimu řízení a slouží pro jednoduché nastavení rychlosti. Požadovaná rychlost se vynuluje přepnutím režimu, nebo dvojitým poklepáním na posuvník. Tato funkcionalita byla doplněna z důvodu bezpečnosti, aby bylo nastavení nulové rychlosti co nejrychlejší.

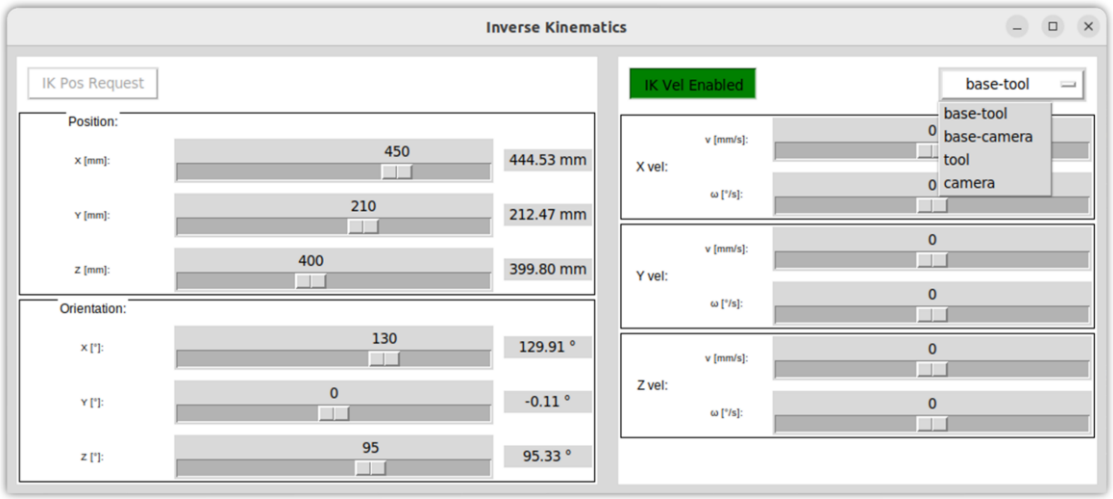
Tlačítka *home* a *zero* slouží pro inicializaci kloubu. *Home* odešle požadavek na service, který zajistí najetí kloubu na inicializační pozici. Ta je reprezentována limitním spínačem. Po stisku tlačítka *zero* kloub najede do pozice, která je označena jako nulová. Tato pozice vychází z CAD modelu. Po stisknutí některého ze zmíněných tlačítek dané tlačítko změní barvu na oranžovou. To značí probíhající proces. Po úspěšném dokončení daného procesu tlačítko zezelená. Proces najetí na nulovou pozici je dostupný až po úspěšné inicializaci kloubu. Než se tak stane, je toto tlačítko deaktivováno. Popsané grafické interakce jsou znázorněny v Obr. 54.



Obr. 54: GUI – Inicializace a nulování kloubu

5.9.3 Kinematika

Panel pro ovládání akcí spojených s kinematikou manipulátoru je otevřen formou pop-up okna stisknutím tlačítka *Inverse Kinematics* na hlavní obrazovce. Jeho rozložení je zobrazeno v Obr. 55.



Obr. 55: GUI – Pop-up kinematiky

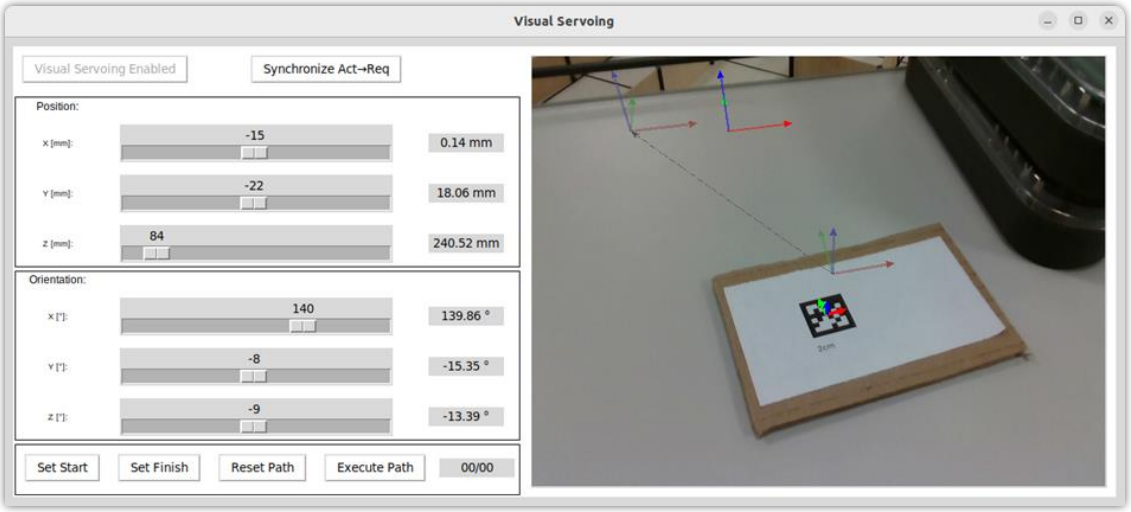
Levá část slouží pro nastavené pózy v pozičním řízení. Pomocí posuvníků je nastavena požadovaná póza. Odeslání požadavku je provedeno tlačítkem *IK Pos Request*, pokud je povoleno, tzn. nacházíme se v pozičním režimu. Vedle každého posuvníku je zobrazena aktuální hodnota. Orientace se v rámci ROS2 Network přenáší ve formě kvaternionů. Tato reprezentace orientace má řadu výhod, není však vůbec intuitivní. V GUI jsou tak kvaterniony transformovány na Eulerovy úhly.

V rychlostním režimu je aktivní pravá část panelu. Zde uživatel dostává možnost volby souřadnicového systému, ve kterém chce pohyby provádět. *Base-tool* znamená rychlost TCP v globálním souřadnicovém systému. Naopak volba *tool* posílá požadavky na pohyb TCP ve vlastním lokálním souř. systému. Totéž platí pro volby *base-camera* a *camera*. Po zvolení požadovaného souř. systému je samostatně nastavena lineární a rotační rychlost kolem každé z os. Požadavky se odesílají pouze pokud je sepnuto tlačítko *IK Vel Enabled* – svítí zeleně. Tyto posuvníky mají taktéž naprogramovanou doplňkovou funkci, která s dvojitým poklepáním nastaví nulovou rychlost.

5.9.4 Vizuální řízení

Vizuální řízení je konfigurováno přes pop-up okno, které je vyvoláno tlačítkem *Visual Servoing* na hlavní obrazovce. Toto okno je znázorněno v Obr. 56.

V rámci této aplikace je manipulátor vybaven dvěma funkcemi spojenými s vizuálním řízením. V prvním režimu manipulátor zajišťuje minimalizaci odchylky mezi aktuální pózou objektu a požadovanou pózou. Jedná se o klasickou úlohu vizuálního řízení. Druhý režim pracuje s plánovačem trajektorie. Předmětem plánování je požadovaná póza, která je postupně upravována podle specifikované trajektorie. To způsobí relativní pohyb mezi TCP a sledovaným objektem. V průběhu vykonávání dané operace se tak objekt může pohybovat a TCP bude vykonávat onen relativní pohyb. Tato vlastnost je výhodná hlavně v dynamických prostředích, kde se poloha objektu mění i v průběhu vykonávání operace.



Obr. 56: GUI – Pop-up vizuálního řízení

V levé části uživatel nastavuje požadovanou pózu objektu v kameře. Taktéž je mu zobrazena aktuální detekovaná póza. Orientace je opět reprezentována Eulerovými úhly. Tlačítko *Synchronize Act→Req* slouží pro synchronizaci aktuální a požadované pózy objektu. Tato funkce byla implementována s cílem zjednodušit uživatelskou interakci. Pokud chceme zadat požadovanou pózu v blízkosti pózy aktuální, tak není nutné všemi posuvníky zdlouhavě nastavovat hodnoty. Stisknutím tlačítka *Synchronize* se všechny posuvníky nastaví na hodnotu reprezentující aktuální pózu objektu.

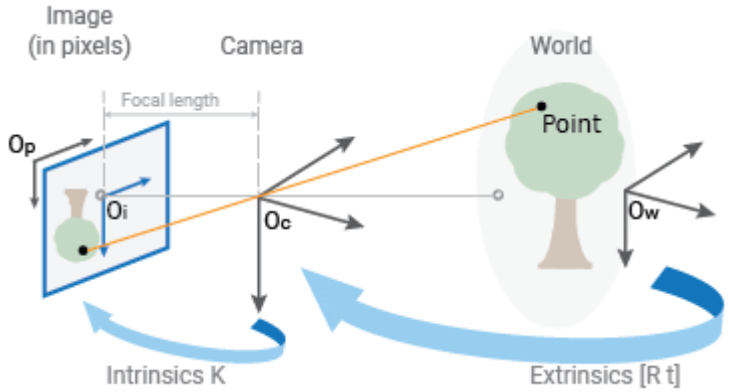
Aktuální snímek z kamery se zobrazuje v pravé části okna. Do tohoto snímku je navíc vykreslen souřadnicový systém objektu pomocí tučných šipek s barvami RGB odpovídajícími ortogonálním osám XYZ. Požadovaná póza je reprezentována čárkovanými šipkami.

Při stisknutí tlačítka *Visual Servoing Enabled* toto tlačítko zezelená a aktivuje se vizuální řízení. Manipulátor se snaží minimalizovat odchylku mezi aktuální detekovanou pózou objektu a pózou žádanou. Zmíněné tlačítko se povoleno, jen pokud je manipulátor v rychlostním režimu. Pokud toto tlačítko aktivní není (nesvítí zeleně), je možné vyvolat vykonání trajektorie mezi počáteční a koncovou pózou značky.

Pro práci s plánovačem trajektorie slouží spodní část okna. Teorii ohledně tohoto plánovače se věnuje kapitola 4.5, praktickému nasazení potom kapitola 7.2. Tlačítka *Set Start* a *Set Finish* slouží pro uložení počáteční a koncové požadované pózy, kterou má objekt zaujmout. Tyto pózy jsou zobrazeny souř. systémy s větší průhledností. Trajektorie mezi těmito pózami je vykreslena šedou šipkou. Při stisku tlačítka *Reset Path* jsou ze snímku uložené pózy smazány. Tlačítko *Execute Path* vyvolá naplánování a vykonání trajektorie mezi počáteční a koncovou pózou objektu v kameře. Aktuální postup při vykonávání naplánovaných kroků je ilustrován v políčku, které je v Obr. 56 nastaveno na hodnotu 00/00. V průběhu vykonávání trajektorie je toto políčko oranžové. Po úspěšném dokončení procesu políčko zezelená.

5.9.5 Vykreslování souřadnicových systémů

Pro vykreslení všech zmíněných souřadnicových systémů vznikla funkce displayFrame. Tato funkce nejprve vytvoří čtveřici bodů ve 3D prostoru. Ty reprezentují počátek a konce os x, y, z – vzniká tedy ortonormální souřadnicový systém v počátku. Pro transformaci tohoto souřadnicového systému do vhodné 3D pózy a jeho vykreslení ve 2D snímku je využit tzv. pinhole projekční model kamery [25] [49], který je popsán rovnicí *(5.36)*. Tato transformace je aplikována na všechny čtyři 3D body uvedeného souřadnicového systému. Výsledné 2D body jsou následně v obraze vhodně propojeny barevnými šipkami.



Obr. 57: Pinhole projekční model [49]

$$\boldsymbol{x}' = \boldsymbol{P} \cdot \boldsymbol{X} = \boldsymbol{K} \cdot \boldsymbol{H} \cdot \boldsymbol{X} \tag{5.36}$$

Kde:

- $\boldsymbol{x}'$ – Souřadnice 2D bodu ve snímku – homogenní
- $\boldsymbol{P}$ – Projekční matice
- $\boldsymbol{K}$ – Vnitřní parametry kamery *(5.38)*
- $\boldsymbol{H}$ – Vnější parametry – matice homogenní transformace – póza souř. systému
- $\boldsymbol{X}$ – Souřadnice bodu ve 3D

Jelikož se ve výpočtech *(5.36)* uplatňují homogenní souřadnice $\boldsymbol{X} = (x, y, z, 1)$, výsledný 2D bod $\boldsymbol{x}' = (x', y', w)$ obsahuje homogenní složku w. Abychom získali čistý 2D bod ve tvaru $\boldsymbol{x} = (x, y)$, je nutné provést dehomogenizaci, která je popsána rovnicí *(5.37)*.

$$\boldsymbol{x} = \left(\frac{x'}{w}, \frac{y'}{w}\right) \tag{5.37}$$

Matice $\boldsymbol{K}$ představuje kalibrační matici kamery, někdy též nazývanou matice intrinsických (vnitřních) parametrů. Její tvar je popsán rovnicí *(5.38)*. Poslední sloupec této matice je nulový. Matice $\boldsymbol{K}$ je běžně rozměru 3x3. Zde je však nutné jeden sloupec doplnit, aby matici bylo možné jednoduše násobit maticí homogenní transformace H, která má rozměr 4x4.

$$\boldsymbol{K} = \begin{pmatrix} f_x & 0 & c_x & 0 \\ 0 & f_y & c_y & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tag{5.38}$$

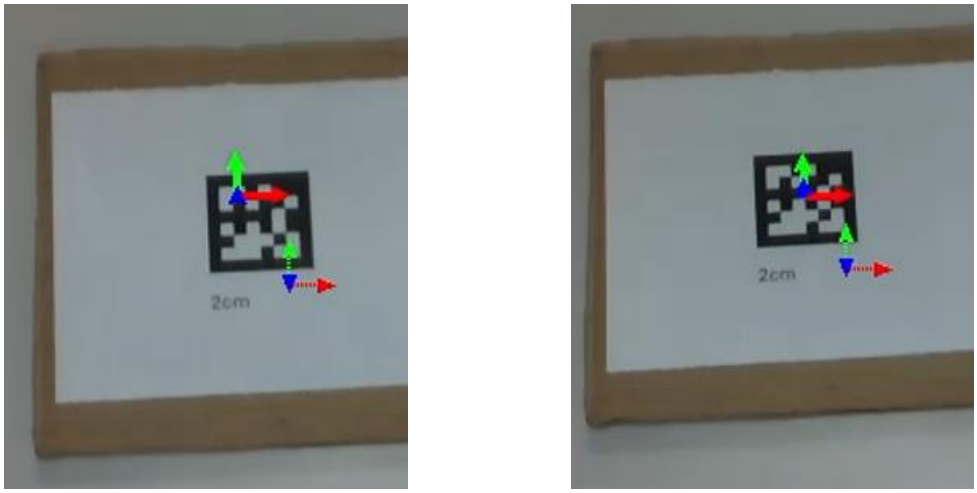
Kde:

- f_x, f_y – Ohniskové vzdálenosti
- c_x, c_y – Hlavní bod = Souřadnice pixelu, kterému odpovídá střed kamery

5.9.6 Integrace ROS

Program grafického rozhraní musí běžet v nekonečné smyčce, aby mohl reagovat na uživatelské akce. K zajištění komunikace se zbytkem systému robotického manipulátoru je nutné tuto smyčku cyklicky přerušovat. Perioda přerušení je nastavena na 20 ms. Vždy po uplynutí této periody je zpracována veškerá komunikace s manipulátorem, která se během této doby ukládala do fronty. Popsaný přístup umožňuje současnou existenci obou smyček bez potřeby vícevláknové synchronizace obrovského množství komunikovaných proměnných. Komunikační rozhraní tohoto uzlu je popsáno v Tab. 11.

Popsaná architektura však zaostává při zobrazování snímků z kamery. Ty jsou přenášeny formou ROS2 zprávy typu Image v rozlišení 640x480p. Snímek je nutné konvertovat do tkinter.Image a vykreslit. Tím vzniká zpoždění, které je viditelné z Obr. 58. Na změnu pózy objektu nejprve reaguje vykreslení odpovídajícího souřadného systému. To totiž zpracovává pouze zprávu PoseStamped, což je v základu sedm reálných čísel. Vykreslení nového barevného snímku však proběhne až s přibližně 100ms zpožděním. Synchronizací snímku a souřadného systému by sice vznikl na první pohled více uniformní obraz, došlo by však k ještě většímu zpoždění. Samotný algoritmus detekce obrazu ani vizuálního řízení samozřejmě zmíněným zpožděním netrpí.



Obr. 58: Zpoždění vykreslování snímků

5.10 Model manipulátoru

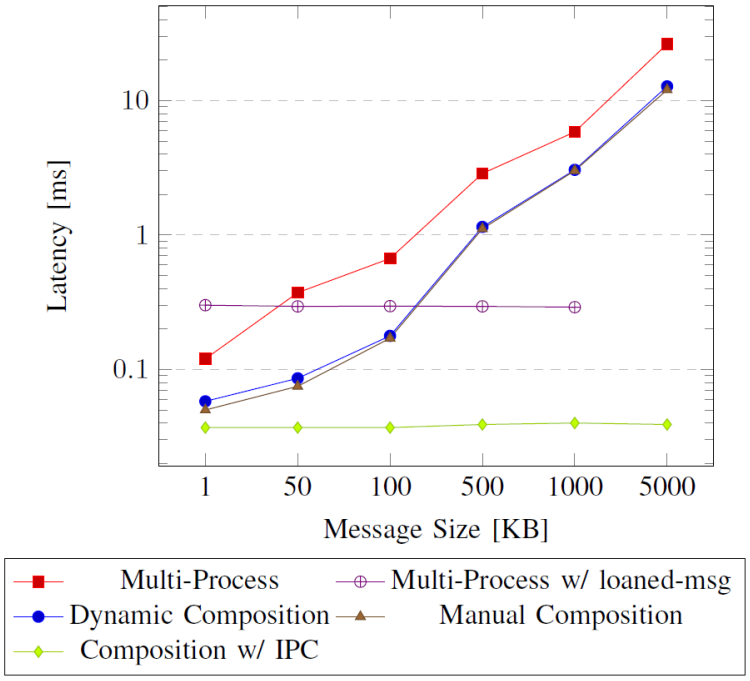
Balík roboarm_desctiption se v rámci aplikace přímo nespouští, nemá totiž žádný executable. Představuje však soubor veškerých informací spojených s mechanikou manipulátoru. Uchovává URDF a XACRO popis manipulátoru, které jsou využity v Kinematics uzlu. Stejně tak obsahuje veškeré CAD modely robotu využívané během vizualizace v RViz.

5.11 Uzlová kompozice a vnitro-procesová komunikace

Uzel ObjectDetection, který se stará o estimaci pózy objektu v kameře, přijímá snímky z kamery formou ROS2 zprávy typu Image. Na základě měření bylo zjištěno, že přenos snímkových dat z uzlu CameraDriver do uzlu ObjectDetection trvá v průměru 500 ms. Samotný proces detekce uvnitř uzlu ObjectDetection přitom zabere přibližně 60 ms. Zpoždění způsobené přenosem obrazu je tak řádově vyšší. Během vizuálního řízení, a řízení obecně, je zásadní co nejvíce omezit zpoždění v systému. Z toho důvodu byl nasazen mechanismus uzlové kompozice, kterému se věnuje tato kapitola.

ROS2 pomáhá s vytvořením modulární a distribuované sítě uzlů. Tyto uzly mohou být vytvořeny s využitím různých programovacích jazyků a mohou být spuštěny na různých zařízeních. Proto je nutné, aby takové uzly používaly standardizovanou komunikaci. Tu zajišťuje middleware s názvem DDS (Data Distribution Service), který specifikuje protokol RTPS (Real Time Publish Subscribe). Tento middleware zajišťuje transportní mechanismy, serializaci dat, objevování uzlů aj. DDS v základním nastavení využívá komunikační protokol UDP v režimu unicast, případně multicast. [50]

Každý uzel je spuštěn jako samostatný proces. Při komunikaci mezi uzly tak každá zpráva musí projít všemi vrstvami UDP stacku i v případě, že uzly běží na stejném zařízení. To vede ke zpoždění, které je úměrné množství přenášených dat, jak ukazuje Obr. 59 a jak se projevilo také v naší aplikaci. Komunikace mezi uzly, které běží v různých procesech je v grafu vyznačena červenou barvou.



Obr. 59: Latence DDS komunikace [51]

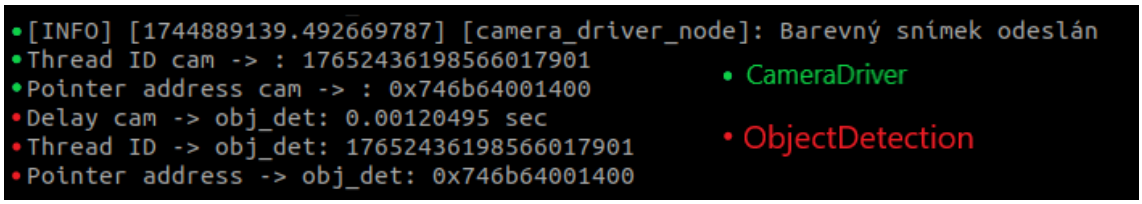
Tento problém by bylo možné vyřešit sloučením uzlů ObjectDetection a CameraDriver do jediného nového ROS2 balíku se společným C++ kódem. Tím by ale

zanikla podstatná výhoda systému ROS2 a tou je modularita. Při potenciální budoucí záměně kamerového systému by bylo nutné přepisovat kód, který je společný i pro detekci objektů. Stejný problém by nastal při nasazení jiného algoritmu pro detekci.

Vhodnějším řešením je integrace mechanismu tzv. **uzlové kompozice** (anglicky Node composition). Jednotlivé uzly nejsou vytvořeny jako samostatné spustitelné procesy, ale jako tzv. komponenty. Tyto komponenty poté skládáme do tzv. kontejnerů. Všechny komponenty uzavřené do stejného kontejneru jsou poté **spuštěny v rámci jediného procesu**. Výrazně tak poklesne latence, zároveň ale není narušena modulární podstata systému. Jednotlivé komponenty jsou stále implementovány jako samostatné ROS2 balíky s odděleným kódem. Popsaný pokles latence znázorňuje v Obr. 59 modrý průběh. V naší aplikaci došlo při uzavření uzlů/komponentů CameraDriver a ObjectDetection do jednoho kontejneru k poklesu latence z 500 ms na 200 ms.

Přestože došlo k více než dvojnásobnému snížení latence, práce na optimalizacích pokračovaly a došlo k implementaci dalšího komunikačního mechanismu. Tím je tzv. **vnitro-procesová komunikace** (anglicky Intra-Process Communication, dále jen IPC). Po nasazení uzlové kompozice jsou sice jednotlivé uzly spouštěny v rámci jediného společného procesu, pro komunikaci ale stále využívají zmíněný middleware DDS. Po implementaci IPC komunikace obchází DDS vrstvu s RTPS protokolem. Zprávy už nejsou posílány přes UDP/IP stack, ale uzly si předávají přímo **ukazatele do operační paměti**. Tím je zajištěna tzv. zero-copy policy, tedy pravidlo nulového kopírování dat, což zajistí další pokles latence. Závislost latence na množství dat s využitím mechanismu IPC je v Obr. 59 znázorněna zelenou barvou.

V rámci této aplikace klesla latence z původních 500 ms na 1.2 ms, jak ukazuje výpis v Obr. 60. **Díky optimalizacím tak došlo k více než stonásobnému zrychlení komunikace mezi uzly**. Změřená latence je však stále řádově vyšší, než říká analýza IPC mechanismu [51]. Uzel ObjectDetection využívá SingleThreadedExecutor. Ten je s největší pravděpodobností zaneprázdněn vykonáváním ostatních funkcí a představuje limitující faktor. Přesná analýza latence je nad rámec této práce. Latence způsobená čistě přenosem mezi uzly by při komunikaci s IPC měla představovat hodnoty v řádech stovek mikrosekund.



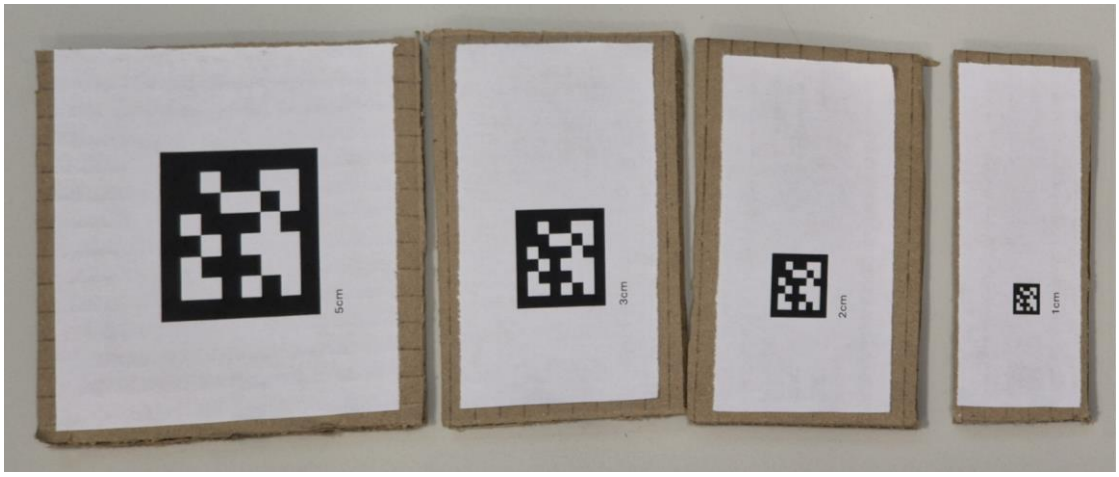
Obr. 60: Výpis komunikace s nasazením IPC

Ve výpisu na Obr. 60 je znázorněno, že komponenty CameraDriver i ObjectDetection běží na totožném vlákně a předávají si totožný ukazatel na data. To značí, že IPC komunikace byla úspěšně zprovozněna.

Na první pohled by se mohlo zdát, že do uzlové kompozice by v rámci potlačení latence v systému měly být zahrnuty také uzly VisualServoing a ManipulatorDriver. Při komunikaci mezi uzly ObjectDetection a VisualServoing však nedochází k přenášení velkého množství obrazových dat. Komunikace zahrnuje jediný datový tok typu PoseStamped, což je v základu sedm reálných čísel. Latence tohoto komunikačního vlákna tak podle měření představuje méně než 2 ms. ManipulatorDriver do uzlové kompozice být zahrnut nemůže z toho důvodu, že tento uzel je vytvořen v jazyce Python. Uzlová kompozice nedovoluje uvnitř jediného kontejneru míchání uzlů vytvořených v jazyce C++ a Python.

6 SPOLEHLIVOST DETEKCE OBJEKTŮ

V rámci práce byly sledované objekty opatřeny značkami typu AprilTag. V průběhu testování byly ověřeny schopnosti rozpoznávat značky o velikosti 10, 20, 30 a 50 mm, jak ukazuje Obr. 61. Výsledná volba vhodné velikosti vychází ze vzdálenosti, ve které je kamera schopná objekty detekovat a z pracovního prostoru manipulátoru.

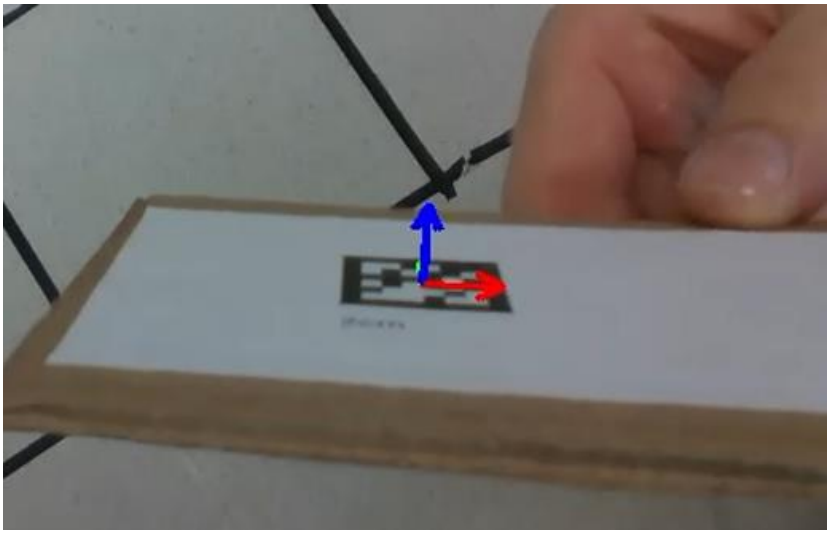


Obr. 61: Testované rozměry AprilTagů

Značka s délkou hrany 10 mm se projevila jako ideální v úlohách sledování dynamických objektů ve velmi blízké vzdálenosti. Díky malým rozměrům značky ji zvládlo vizuální řízení spolehlivě udržel v zorném poli kamery. Při vzdálenostech nad 300 mm však detektor ve snímku s rozlišením 640x480p značku nedokázal spolehlivě vyhledat. Pro účely této aplikace je tak nevyhovující.

Naopak detekce 30mm a 50mm značek byla velmi spolehlivá ve větších vzdálenostech. Detektor zvládal pózy značek vyhledat ve vzdálenostech přesahujících 100 cm. Tato schopnost však pro naši aplikaci nemá žádný praktický význam. V blízkých vzdálenostech značky těchto velikostí nebyly použitelné. Vlivem omezené dynamiky manipulátoru se stávalo, že s jakýmkoliv pohybem značka opustila poměrné nízké zorné pole kamery a vizuální řízení se z bezpečnostních důvodů přerušilo.

Jako nejvíce univerzální se projevila značka s délkou hrany 20 mm. Značka byla spolehlivě detekována ve vzdálenostech do 550 mm při přímém pohledu. Při vyklonění roviny značky do úhlu 75° vzhledem k rovině kamery se vzdálenost spolehlivé detekce snížila na 450 mm. Maximální vyklonění značky, které je detektor schopný zachytit je vyobrazeno v Obr. 62.

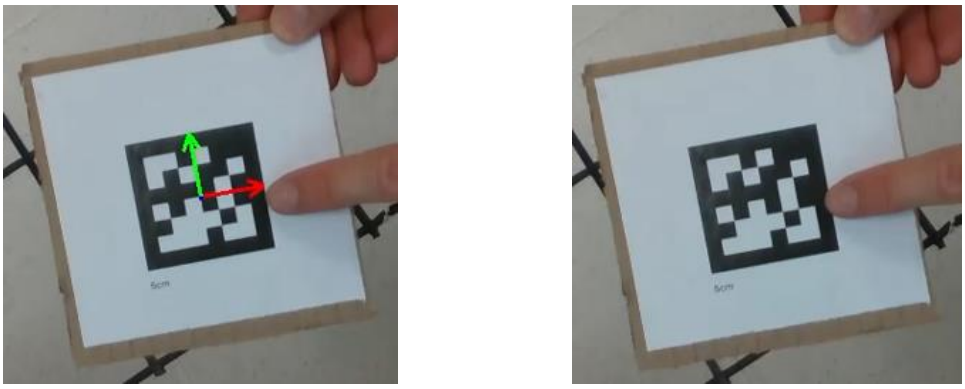


Obr. 62: Maximální detekovatelné vyklonění značky

20mm značka je stále dostatečně malá na to, aby se celá vešla do zorného pole kamery i ve velmi blízkých vzdálenostech představujících desítky milimetrů. Proto byla právě tato velikost značky zvolena pro všechny další operace týkající se vizuálního řízení.

Světelné podmínky na úlohu detekce neměly podstatný vliv. Přesné měření osvětlení místnosti je za hranicí rozsahu této práce. Detektor byl však schopen rozpoznat značky i v podprůměrně osvětlené místnosti. AprilTag představuje robustní typ binární značky, která je velmi dobře detekovatelná i za nepříznivých podmínek. Proto se také v robotice hojně využívá.

Použitý detektor však výrazně zaostává v případě, kdy ve snímku nemá dostupnou zcela kompletní značku. Tato situace nastane při překrytí značky jiným objektem, nebo když značka vlivem omezené dynamiky manipulátoru opustí zorné pole kamery. Detektor selhává už při minimálním překrytí, jak ukazuje Obr. 63.



Obr. 63: Selhání detektoru při překrytí značky

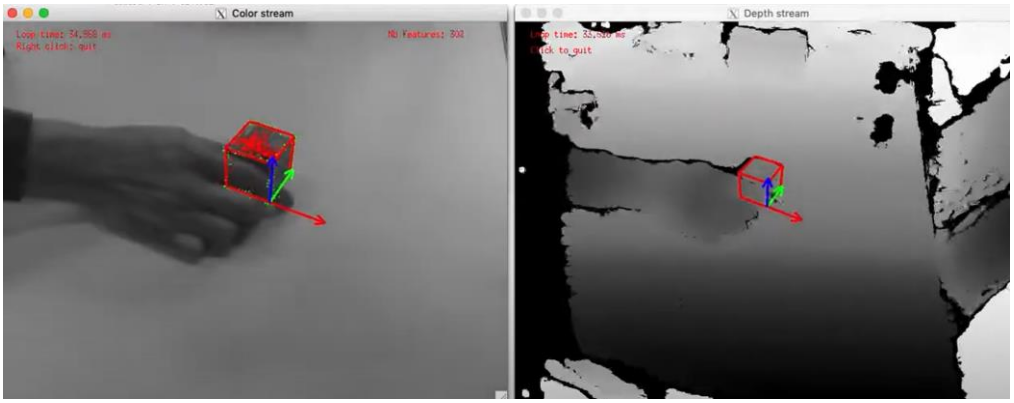
7 DISKUSE

Vlivem neočekávaných komplikací spojených s konstrukcí manipulátoru nebylo možné realizovat některé doplňkové cíle, které byly původně plánovány nad rámec základního zadání. Hlavní cíle projektu, definované v zadání, však úspěšně splněny byly.

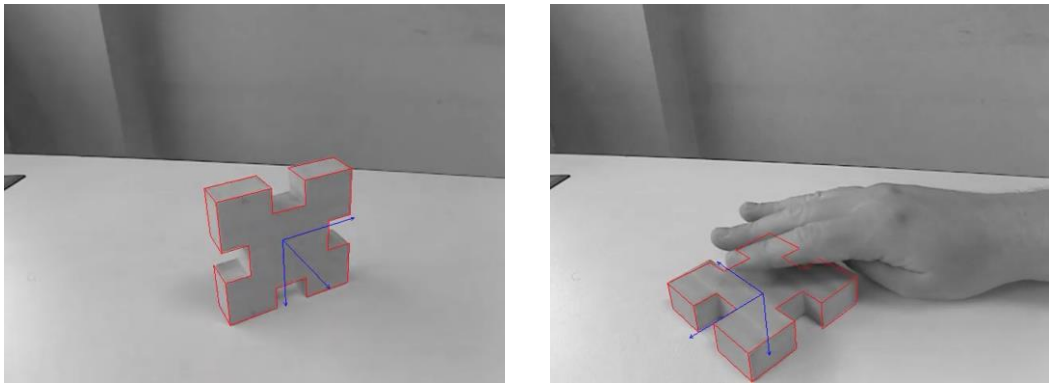
7.1 Detekce objektů na základě CAD modelů

V současné době jsme schopni detekovat objekty, které jsou opatřeny značkou typu AprilTag. S cílem zvýšení univerzality celé úlohy je vhodné nasadit detektor, který bude schopen detekovat objekty a estimovat jejich pózu i bez předchozího označení. Na toto téma byla provedena rešerše a byl vyvozen závěr, že do našeho systému by bylo vhodné integrovat detektor poskytovaný platformou ViSP, stejně jako stávající nasazený detektor AprilTag značek. Jedná se o tzv. markerless model-based tracker. Takový detektor rozpoznává objekty, které nejsou opatřeny žádnou lokalizační značkou. Toho dosahuje na základě modelu daného objektu, který je před samotnou detekcí nutné specifikovat. Modely jsou popisovány ve formátu VRML (Virtual Reality Modeling Language). Jedná se o textový formát pro popis geometrie 3D objektů. Konverzi mezi klasickými STL soubory a formátem VRML zajišťují různé programy pro tvoru modelů (např. Blender), případně jsou dostupné online převodníky (např. AnyConv).

Detektor zpracovává současně data z barevného a z hloubkového kanálu RGB-D kamery. Obsahuje nativní podporu kamer řady Intel RealSense, do které spadá kamera využitá v této aplikaci. Při rozpoznávání objektů se uplatňuje kombinace více vizuálních rysů extrahovaných jak z barevných, tak z hloubkových dat. Nad barevnými daty pracuje detektor významných bodů a pohybujících se hran. Normály ploch a mračno bodů jsou získány z hloubkových dat. Demonstrace takového detektoru je ilustrována v Obr. 64. Použitá kamera Intel RealSense D435 nedokáže měřit vzdálenost příliš blízkých objektů. V takovém případě se hloubková data neuplatní a detektor pracuje s čistě barevnými daty. I v tomto stavu však podle dostupných ukázek a měření poskytuje uspokojivé výsledky, jak ukazuje Obr. 65. [52] [53] [54]



Obr. 64: RGB-D detektor objektů [52]



Obr. 65: RGB detektor objektů [55]

Díky modulární struktuře programu, který je postavený na ROS2 architektuře bude proces integrace nového detektoru jednoduchý. Balík CameraDriver je už nyní schopen odesílat kompletní RGB-D data. Hloubkový kanál je v současnosti deaktivován, neboť jej stávající detektor nevyužívá. Nový balík ObjectDetection bude mít v rámci ROS2 Network totožný komunikační interface. Bude stále poskytovat pózu objektu ve formátu PoseStamped a okolní uzly tak nebudou nijak ovlivněny.

7.2 Plánovač trajektorie

V současnosti je do systému integrován jednoduchý plánovač, který počítá trajektorii s využitím algoritmů LERP a SLERP. Výstupem je trajektorie, která zajišťuje plynulou změnu polohy a orientace lineárním způsobem. Aby bylo možné manipulátorem provádět složitější operace nad sledovaným objektem, bylo by vhodné nasadit pokročilejší plánovač. Ten by generoval komplikovanější trajektorie mezi více než dvěma body. Vylepšený plánovač by také generoval kruhové, případně spline trajektorie.

7.3 Volba kamery

Tato aplikace využívá RGB-D kameru Intel RealSense D435, která poskytuje jak barevné snímky, tak hloubková data. Nízké zorné pole (Field of View – dále jen FOV) barevné kamery však představuje jisté omezení. V průběhu testování se stávalo, že sledovaný objekt opustil FOV kamery a vizuální řízení bylo přerušeno. Jedním z možných řešení je integrace jiného typu kamery s vyšším FOV. Tímto bychom však potenciálně přišli o hloubková data. V současnosti je využit detektor, který rozpoznává objekty opatřené AprilTag značkami a hloubkový kanál se nevyužívá. V kapitole 7.1 je však představen návrh nového detektoru, který s RGB-D daty pracuje. V případě záměny kamery by tak nebylo možné jej nasadit.

Popsaný problém ztráty objektu plyne z nízkých dynamických vlastností použitého manipulátoru. Při eliminaci problémů konstrukce se tak dynamika zvýší a manipulátor bude schopen sledovat objekt i v případě menšího FOV a kamera Intel RealSense D435

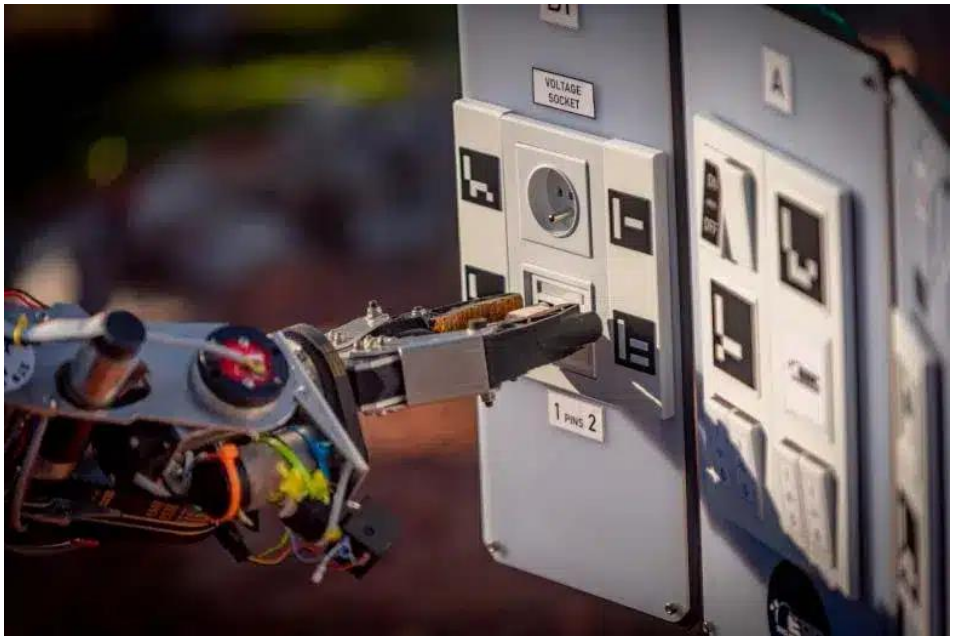
tak bude vyhovující. Finální rozhodnutí o záměně kamery tak závisí na požadavcích na konkrétní aplikaci a na dynamických vlastnostech manipulátoru.

7.4 Brno Mars Rover

Důvodem vzniku této aplikace je následné nasazení v rámci projektu studentského týmu Brno Mars Rover. Cílem je vývoj plně autonomního mobilního robotického systému určeného pro simulovaný průzkum Marsu v rámci mezinárodní soutěže European Rover Challenge (dále jen ERC). Tým sdružuje znalosti z oblasti návrhu mechanické konstrukce, autonomní navigace, robotické manipulace, bezpilotních letounů aj. Popsaný systém je zobrazen v Obr. 66. [56]

Obr. 66: Rover s označením Freya [56]

V rámci ECR bude naše aplikace uplatněna v řadě definovaných úloh. Manipulátor bude, s využitím kamerového systému, rozpoznávat vzorky hornin, které posbírá a vloží do speciálního kontejneru pro budoucí analýzu. Při vykonávání další úlohy bude manipulátor ovládat elektrický panel, který byl původně určen pro lidskou interakci a nemá žádný robotický interface, jak znázorňuje Obr. 67. V obou zmíněných případech manipulátor uplatní veškeré schopnosti vytvořené v rámci této práce.



Obr. 67: ERC – elektrický panel [57]

7.4.1 Vylepšená konstrukce manipulátoru

Aktuální stav konstrukce manipulátoru je pro nasazení v ERC nevyhovující. Proto aktuálně vzniká nová a vylepšená verze. Díky tomu, že pro popis kinematického řetězce je v této práci využit formát URDF, stává se celé řešení univerzálním. Pro nasazení na novém robotickém rameni tak bude nutné modifikovat URDF soubor a zbytek programu může zůstat nezměněn. Univerzálnost řešení podporuje také integrace sytému ROS. Ten je modulární a například driver serv manipulátoru a kamery může být jednoduše nahrazen za kód pro jiný hardware. Při návrhu konstrukce nového manipulátoru jsou uplatňovány veškeré znalosti, z oblasti mechaniky a kinematiky, získané řešením problémů popsaných v kapitole 1.3.

8 ZÁVĚR

V rámci této práce byl úspěšně navržen a implementován systém pro řízení šestiosého manipulátoru za pomoci vizuální zpětné vazby. Zmíněným manipulátorem bylo robotické rameno Arctos ve verzi V0.2. Jedná se o open-source projekt šestiosého robotického ramene, které svou strukturou připomíná lidskou paži. Tento typ konstrukce je univerzální a vhodný pro řadu různých aplikací. Manipulátor Arctos je téměř kompletně vyroben z plastu metodou 3D tisku. Díky tomu je velmi dostupný, 3D tisk však přináší řadu komplikací. Jak už napovídá označení verze, manipulátor se stále nachází ve fázi vývoje. Jeho konstrukce disponuje řadou problémů, které se v průběhu této práce projevovaly velmi často a vývoj řídicí aplikace mnohdy kompletně zablokovaly.

Řešení konstrukčních problémů nebylo součástí zadání, jejich opakovaný výskyt měl však zásadní dopad na průběh vývoje řídicí aplikace. Manipulátor nebyl schopen vykonávat plynulé pohyby, převodová ústrojí kloubů se také často kompletně zadrhla a bylo nutné je manuálně roztočit. Eliminace zmíněných problémů byla tedy nevyhnutelná, spotřebovala však podstatné množství času určeného pro vývoj samotné řídicí aplikace. Část manipulátoru byla několikrát rozložena a promazána s mírným dočasným zlepšením stavu. Konstrukci diferenciálního zápěstí robotu bylo posléze nutné znovu vytisknout s vyšší přesností, což vedlo k částečnému potlačení problémů. Manipulátor stále není schopen vykonávat zcela plynulé pohyby, má výraznou vůli v převodech, konstrukce není dostatečně tuhá a při jakémkoliv pohybu se rozkmitá. Robotické rameno taktéž není schopno vykonávat velmi pomalé pohyby, které jsou zásadní při precizním vizuálním řízení. Stejně tak nezvládá velmi rychlé pohyby, které jsou při vizuálním řízení zásadní. Často také dochází k prokluzu řemene hned u několika kloubů, což vede ke ztrátě informace o aktuální poloze koncového bodu manipulátoru. V rámci práce byly identifikovány hlavní zdroje popsaných problémů. Znalosti získané identifikací a řešením konstrukčních problémů jsou využity při vývoji nové verze manipulátoru, která aktuálně probíhá.

Mezi první kroky při návrhu systému patřil vývoj demonstrační aplikace pro testovací účely a seznámení se se způsoby řízení manipulátoru. Vznikla první iterace jednoduchého grafického rozhraní, které výrazně urychlilo proces identifikace parametrů manipulátoru. Jako výchozí stav práce byl předán ovladač kloubových pohonů manipulátoru, který byl považován za hotové řešení. Ukázalo se však, že využívá nekorektní přístup ke komunikační sběrnici s kloubovými pohony. To způsobovalo nezanedbatelné zpoždění a tento ovladač se ukázal jako nevyhovující. Jediný kus obdrženého kódu tak bylo nutné od základů přepracovat. Nad rámec zadání tak vznikl vícevláknový asynchronní driver, jehož nasazením došlo k téměř stonásobnému zrychlení komunikace. Při vícevláknovém programování se začaly projevovat problémy plynoucí z nevhodného návrhu vícevláknového executoru, který v ROS2 systému řídí přiřazování zdrojů jednotlivým

funkcím. Problém s tzv. neférovým plánováním byl detailně prozkoumán, popsán a následně potlačen.

Struktura robotického ramene je popsána kinematickým modelem, který byl vytvořen jednak formou Denavit-Hartenberg parametrů a jednak formou URDF popisu. Byla popsána úloha přímé kinematiky, která podle známých natočení kloubů spočítá polohu a orientaci koncového bodu manipulátoru v kartézských souřadnicích. Rovněž byly diskutovány přístupy k řešení inverzní úlohy kinematiky, která řeší opačný problém a je podstatně složitější. Z výčtu nástrojů dostupných v rámci ROS2 systému byla pro své vlastnosti nasazena knihovna Orocos KDL. Pro dlouhodobě účinné řešení rychlostní inverzní kinematiky byl nasazen pokročilý solver, s definovanou sekundární úlohou kinematiky, potlačující redundantní rotace kloubů vlivem singularit s využitím nullspace optimalizace. Algoritmus použitého solveru byl detailně popsán.

Byly diskutovány a porovnány metody vizuálního řízení společně s různými typy kamerových systémů a dopady jejich umístění v systému robotu. Na základě analýzy systému a vlastností jednotlivých přístupů byla v rámci této aplikace implementována metoda vizuálního řízení založeného na pozici (Position-Based Visual Servoing – PBVS) s využitím platformy ViSP. Jako kamerový systém byla nasazena hloubková kamera (RGB-D) Intel RealSense D435 v konfiguraci eye-in-hand. Bylo vytvořeno konstrukční řešení uchycení kamery přímo na robotickém rameni v blízkosti nástroje. To jí zajišťuje manévrovatelnost a možnost prohledávat scénu z různých úhlů. Toto řešení přináší nevýhodu projevující se v situaci, kdy robotický systém není dostatečně dynamický, což je náš případ. Manipulátor není schopen provádět dostatečně rychlé pohyby a hrozí ztráta sledovaného objektu ze zorného pole kamery. V rámci práce byly navrženy způsoby, jak tuto nevýhodu potlačit. Byla také popsána a provedena hand-eye kalibrace, která poskytuje přesnou homogenní transformaci mezi kamerou a koncovým bodem manipulátoru. Tato kalibrace je nutná pro zajištění precizního řízení. Vizuální řízení do jisté míry potlačuje problémy spojené s nedokonalostí konstrukce. V rámci regulační smyčky představují hardwarové nepřesnosti regulační poruchu u, kterou je regulátor do jisté míry schopen kompenzovat. Preciznost nastavení požadované pózy objektu je však omezena skutečností, že robotické rameno není schopno vykonávat dostatečně jemné pohyby. Na druhé straně dynamika řízení je omezena nízkou tuhostí konstrukce. Už při poměrně pomalých pohybech se manipulátor rozkmitá a hrozí jeho poškození.

Klasická úloha vizuálního řízení umožňuje manipulátoru sledovat objekt v prostoru. Pokud ale s objektem chceme nějakým způsobem interagovat, je nutné využít plánovač trajektorie. V rámci této práce byl implementován jednoduchý offline plánovač lineární trajektorie metodou LERP a SLERP, který generuje sekvenci požadovaných póz. Plánovač však není nasazen obvyklým způsobem, kdy by naslepo řídil přímo pózu koncového bodu manipulátoru v prostoru. Místo toho spolupracuje s algoritmem PBVS, kterému vytvořenou sekvenci postupně předává jako požadovanou pózu objektu v kameře. Plánovaná trajektorie tak neřídí absolutní pózu manipulátoru v prostoru, ale

jeho relativní pózu vzhledem k objektu. Díky tomu je možné interagovat s pohybujícími se objekty, teoreticky do nich například vrtat (v případě lineární trajektorie), bez hrozby poškození nástroje či obrobku.

V rámci této práce je možné sledovat objekty, které jsou opatřeny značkou typu AprilTag. Spolehlivost této detekce byla ověřena a popsána. Použitý detektor zvládá spolehlivě určovat přesnou polohu a orientaci značek v uspokojivém rozsahu. Silně však zaostává při překrytí značky jiným objektem. Proběhla rešerše algoritmů na detekci neoznačených objektů na základě jejich 3D CAD modelu. Vlivem podstatné časové ztráty při řešení konstrukčních problémů však již nedošlo k praktické implementaci. Zadání práce bylo úspěšně splněno, neboť detekce na základě 3D modelu představovala osobní cíl a bude doplněna v rámci budoucích vylepšení.

Při řízení se obecně snažíme minimalizovat časovou prodlevu v regulační smyčce. V této aplikaci to platí rovněž. Při využití klasické komunikace mezi členy ROS2 systému vzniká při přenosu většího množství dat nezanedbatelná latence. Přenos obrazových dat mezi kamerou a detektorem objektu představoval zpoždění v průměru 500 ms. To je v kontextu řízení nepřípustné. Proběhla tedy rešerše pro pochopení způsobu komunikace v ROS2 systému a možností optimalizačních technik. Optimalizace latence v této práci staví na implementaci pokročilých komunikačních mechanismů jako jsou uzlová kompozice (node composition) a vnitro-procesní komunikace (intra-process communication). Jejich nasazením došlo k více než stonásobnému potlačení latence při přenosu obrazových dat z původních 500 ms na 1,2 ms. Teoretické zrychlení je ještě řádově vyšší, provedené měření však omezoval ROS2 executor a rozsáhlejší analýza je mimo rozsah práce.

Rovněž vzniklo uživatelsky přívětivé grafické rozhraní, které slouží pro vizualizaci stavu systému a veškerých provozních veličin. Uživateli dává možnost provádět jednoduchou konfiguraci veškerých parametrů vizuálního řízení, zadávat povely pro pohybové akce manipulátoru, případně ovládat přímo jednotlivé klouby či generovat trajektorie.

Poslední kapitola hovoří o praktickém nasazení této aplikace v projektu Brno Mars Rover v rámci soutěže European Rover Challenge. Pro soutěž je však aktuální konstrukce manipulátoru nevyhovující. Proto momentálně vzniká vylepšená verze, při jejímž vývoji jsou uplatněny veškeré znalosti získané v průběhu této práce.

Navzdory veškerým obtížím spojeným s technickými nedostatky použitého manipulátoru se podařilo systém vizuálního řízení úspěšně navrhnout a implementovat. Robotické rameno dokáže detekované objekty konzistentně sledovat v prostoru a vykonávat nad nimi definované relativní trajektorie. Výsledky práce tak lze v kontextu zmíněných podmínek považovat za pozitivní a představují pevný základ pro další rozvoj. Funkce systému jsou demonstrovány ve formě videodokumentace v příloze a na odkazu [58].

LITERATURA

[1] ARCTOS ROBOTICS, 2024. *Arctos Robotics*. Online. Dostupné z: https://arctosrobotics.com/. [cit. 2024-09-10].

[2] MAKERBASE, 2023. *Makerbase CANable V2.0 User Manual*. Dostupné z: https://github.com/makerbase-mks/CANable-MKS/tree/main. [cit. 2024-06-25].

[3] MAKERBASE, 2024. *MKS SERVO42/57D CAN User Manual V1.0.5*. Dostupné z: https://github.com/makerbase-motor/MKS-SERVO57D/blob/master/User%20Manual/MKS%20SERVO42%2657D_CAN%20User%20Manual%20V1.0.5.pdf. [cit. 2024-06-20].

[4] LUNENBURG, J.J.M.; CLEPHAS, T.T.G.; DIRKX, N.J.; WILLEMS, B.; ELFRING, J. et al., 2011. Tech United Eindhoven Team Description 2011. Online. In: *Proc. CD of the 15th RoboCup International Symposium*. Dostupné z: https://www.researchgate.net/publication/233881989_Tech_United_Eindhoven_Team_Description_2011. [cit. 2024-12-15].

[5] SKAŘUPA, Jiří, 2007. *Průmyslové roboty a manipulátory*. VŠB – Technická univerzita Ostrava: Ediční středisko VŠB – TUO. ISBN 978-80-248-1522-0.

[6] BRANDSTÖTTER, Mathias; ANGERER, Arthur a HOFBAUR, Michael, 2014. An Analytical Solution of the Inverse Kinematics Problem of Industrial Serial Manipulators with an Ortho-parallel Basis and a Spherical Wrist. Online. In: *Proceedings of the Austrian Robotics Workshop 2014*. Linz. Dostupné z: https://www.researchgate.net/publication/264212870_An_Analytical_Solution_of_the_Inverse_Kinematics_Problem_of_Industrial_Serial_Manipulators_with_an_Ortho-parallel_Basis_and_a_Spherical_Wrist. [cit. 2024-11-25].

[7] CRAIG, John J., 2022. *Introduction to Robotics: Mechanics and Control*. 4. vydání, Global Edition. Pearson Education Limited. ISBN 978-1-292-16495-9.

[8] FORMANT.IO, 2024. *URDF*. Online. Dostupné z: https://formant.io/resources/glossary/urdf/. [cit. 2024-12-23].

[9] LEE, C.S.G a ZIEGLER, M., 1984. Geometric Approach in Solving Inverse Kinematics of PUMA Robots. Online. *IEEE Transactions on Aerospace and Electronic Systems*. Roč. 20, č. 6, s. 695–706. Dostupné z: https://ieeexplore.ieee.org/document/4103975. [cit. 2024-12-25].

[10] PICKNIK ROBOTICS, 2024. *MoveIt*. Online. Dostupné z: https://moveit.ai/. [cit. 2024-11-12].

[11] PICKNIK ROBOTICS, 2024. *MoveIt 2 Documentation*. Online. Dostupné z: https://moveit.picknik.ai/main/index.html. [cit. 2024-11-12].

[12] PICKNIK ROBOTICS, 2024. *MoveIt 2 – IKFast Tutorial (ROS 2 Humble)*. Online. Dostupné z: https://moveit.picknik.ai/humble/doc/examples/ikfast/ikfast_tutorial.html. [cit. 2024-11-25].

[13] OROCOS PROJECT, 2024. *Orocos Kinematics and Dynamics Library (KDL)*. Online. Dostupné z: https://orocos.org/kdl.html. [cit. 2024-12-23].
[14] OROCOS PROJECT, 2024. *Orocos KDL Documentation – Overview*. Online. Dostupné z: https://docs.orocos.org/kdl/overview.html. [cit. 2024-12-23].
[15] OPEN ROBOTICS. *Kdl_parser*. Online. ROS Wiki. Dostupné z: https://wiki.ros.org/kdl_parser. [cit. 2025-01-10].
[16] LOPEZ MUÑOZ, Gloria Liliana; SANTOS TORRES DA MOTTA, José Mauricio, 2014. Comparative Performance Analysis of Image-Based and Position-Based Visual Servoing in a 6 DOF Manipulator. Online. In: *ABCM Symposium Series in Mechatronics*. 6. University of Brasília. Dostupné z: https://www.abcm.org.br/upload/files/PI_IV_07%281%29.pdf. [cit. 2025-04-21].
[17] CONG, Vo Duy a HANH, Le Duc, 2023. A review and performance comparison of visual servoing controls. Online. *International Journal of Intelligent Robotics and Applications*. Roč. 7, s. 65–90. Dostupné z: https://doi.org/https://doi.org/10.1007/s41315-023-00270-6. [cit. 2024-10-20].
[18] CHAUMETTE, François a HUTCHINSON, Seth. Visual Servo Control. Part I: Basic Approaches. Online. *IEEE Robotics & Automation Magazine*. Roč. 13, č. 4, s. 82-90. Dostupné z: https://ieeexplore.ieee.org/document/4015997. [cit. 2024-10-11].
[19] HUTCHINSON, Seth; HAGER, Gregory D. a CORKE, Peter I., 1996. A Tutorial on Visual Servo Control. Online. *IEEE Transactions on Robotics and Automation*. Roč. 12, č. 5, s. 651-670. Dostupné z: https://doi.org/10.1109/70.538972. [cit. 2024-10-11].
[20] INRIA, 2020. *ViSP 3.3.0 Tutorial – IBVS with Franka Emika Panda*. Online. Visual Servoing Platform. Dostupné z: https://visp-doc.inria.fr/doxygen/visp-3.3.0/tutorial-franka-ibvs.html. [cit. 2024-12-27].
[21] INRIA, 2020. *ViSP 3.3.0 Tutorial – PBVS with Panda 7-dof robot from Franka Emika*. Online. Visual Servoing Platform. Dostupné z: https://visp-doc.inria.fr/doxygen/visp-3.3.0/tutorial-franka-pbvs.html. [cit. 2024-12-27].
[22] LU, C.-P.; MJOLSNESS, E. a HAGER, G.D., 1996. Online computation of exterior orientation with application to hand-eye calibration. Online. *Mathematical and Computer Modelling*. Roč. 24, č. 5-6, s. 121-143. Dostupné z: https://www.sciencedirect.com/science/article/pii/0895717796001185. [cit. 2024-12-27].

[23] LIPPIELLO, Vincenzo; SICILIANO, Bruno a VILLANI, Luigi, 2005. Eye-in-Hand/Eye-to-Hand Multi-Camera Visual Servoing. Online. In: *Proceedings of the 44th IEEE Conference on Decision and Control and the European Control Conference*. Seville, Spain, s. 5354–5359. Dostupné z: https://www.researchgate.net/publication/224627607_Eye-in-HandEye-to-Hand_Multi-Camera_Visual_Servoing. [cit. 2024-12-26].

[24] FLANDIN, Grégory; CHAUMETTE, François a MARCHAND, Eric, 2000. Eye-in-hand / Eye-to-hand Cooperation for Visual Servoing. Online. In: *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation*. 3. IEEE, s. 2741-2746. Dostupné z: https://ieeexplore.ieee.org/document/846442. [cit. 2024-10-21].

[25] JANÁKOVÁ, Ilona, 2024. *Počítačové vidění – Optické 3D měření*. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií. Dostupné také z: http://vision.uamt.feec.vutbr.cz/POV/lectures/12_Opticke_3D_mereni.pdf. [cit. 2024-09-20].

[26] MOUSER ELECTRONICS. Intel 82635D435IDK5P. Online. Mouser Electronics. Dostupné z: https://cz.mouser.com/ProductDetail/Intel/82635D435IDK5P?qs=PqoDHHvF64%252B5RxJ3DyHfIg%3D%3D&mgh=1. [cit. 2024-09-11].

[27] SERVI, M.; PROFILI, A.; FURFERI, R. a VOLPE, Y. Comparative Evaluation of Intel RealSense D415, D435i, D455, and Microsoft Azure Kinect DK Sensors for 3D Vision Applications. Online. *IEEE Access*. Roč. 12, s. 111311–111321. Dostupné z: https://doi.org/10.1109/ACCESS.2024.3441238. [cit. 2024-09-10].

[28] INRIA, 2024. *Visual Servoing Platform*. Online. Dostupné z: https://visp-doc.inria.fr/doxygen/visp-daily/index.html. [cit. 2024-10-20].

[29] COMPUTER AIDED MEDICAL PROCEDURES & AUGMENTED REALITY GROUP. *Hand-Eye Calibration*. Online. Technische Universität München (TUM). Dostupné z: https://campar.in.tum.de/Chair/HandEyeCalibration. [cit. 2024-12-30].

[30] INRIA, 2024. *Tutorial: Camera eye-to-hand extrinsic calibration*. Online. Visual Servoing Platform. Dostupné z: https://visp-doc.inria.fr/doxygen/visp-daily/tutorial-calibration-extrinsic.html. [cit. 2024-12-30].

[31] INTEL. *Intel® RealSense™*. Online. GitHub. Dostupné z: https://github.com/IntelRealSense. [cit. 2024-12-30].

[32] Off-Line and On-Line Trajectory Planning, 2015. Online. In: *Motion and Operation Planning of Robotic Systems*. Springer International Publishing Switzerland 2015, s. 29–59. Dostupné z: https://doi.org/10.1007/978-3-319-14705-5_2. [cit. 2025-04-05].

[33] MATHWORKS STUDENT COMPETITIONS TEAM, 2019. *Trajectory Planning for Robot Manipulators*. Online. THE MATHWORKS, INC. MathWorks. Dostupné z: https://www.mathworks.com/matlabcentral/fileexchange/71130-trajectory-planning-for-robot-manipulators. [cit. 2025-04-05].

[34] OPEN ROBOTICS, 2025. *Tutorials: Beginner CLI Tools*. Online. ROS 2 Documentation. Dostupné z: https://docs.ros.org/en/foxy/Tutorials/Beginner-CLI-Tools.html. [cit. 2025-04-15].

[35] OPEN ROBOTICS, 2022. *Sensor_msgs*. Online. ROS Wiki. Dostupné z: https://wiki.ros.org/sensor_msgs. [cit. 2025-04-10].

[36] OPEN ROBOTICS, 2022. *Geometry_msgs*. Online. ROS Wiki. Dostupné z: https://wiki.ros.org/geometry_msgs. [cit. 2025-04-10].

[37] OPEN ROBOTICS, 2022. *Nav_msgs*. Online. ROS Wiki. Dostupné z: https://wiki.ros.org/nav_msgs. [cit. 2025-04-10].

[38] OPEN ROBOTICS, 2022. *Std_srvs*. Online. ROS Wiki. Dostupné z: https://wiki.ros.org/ std_srvs. [cit. 2025-04-10].

[39] DZYMFARDREAMER, 2024. *MKS-Servo CAN Interface Library*. Online. GitHub. Dostupné z: https://github.com/DzymFardreamer/mks-servo-can. [cit. 2024-06-10].

[40] TEPER, Harun; KUHSE, Daniel; GÜNZEL, Mario; VON DER BRÜGGEN, Georg; HOWAR, Falk et al., 2024. Thread Carefully: Preventing Starvation in the ROS 2 Multithreaded Executor. Online. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*. Roč. 43, č. 11, s. 3588–3599. Dostupné z: https://www.researchgate.net/publication/385612092_Thread_Carefully_Preventing_Starvation_in_the_ROS_2_Multithreaded_Executor. [cit. 2025-04-21].

[41] SOBHANI, Hoora; CHOI, Hyunjong a KIM, Hyoseung, 2024. Timing Analysis and Priority-driven Enhancements of ROS 2 Multi-threaded Executors. Online. *ArXiv*. Dostupné z: https://arxiv.org/pdf/2408.08440. [cit. 2025-03-20].

[42] SICILIANO, Bruno a KHATIB, Oussama (ed.), 2016. *Springer Handbook of Robotics*. 2nd Edition. Berlin Heidelberg: Springer-Verlag. ISBN 978-3-319-32550-7.

[43] FLACCO, Fabrizio; DE LUCA, Alessandro a KHATIB, Oussama, 2012. Prioritized Multi-Task Motion Control of Redundant Robots under Hard Joint Constraints. Online. *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. Roč. 2012, s. 3970-3977. Dostupné z: https://ieeexplore.ieee.org/document/6385619. [cit. 2025-02-21].

[44] THE OROCOS PROJECT, 2025. *Orocos – Kinematics And Dynamics Library*. Online. GitHub. Dostupné z: https://github.com/orocos/orocos_kinematics_dynamics/tree/master/orocos_kdl/src. [cit. 2025-04-21].

[45] LYNCH, Kevin M. a PARK, Frank C., 2017. *Modern Robotics: Mechanics, Planning, and Control*. Cambridge: Cambridge University Press. ISBN 978-1107156302.
[46] SPONG, Mark W.; HUTCHINSON, Seth a VIDYASAGAR, M., 2005. *Robot Modelling and Control*. John Wiley. ISBN 978-0471649908.
[47] INRIA, 2025. *VpAdaptiveGain*. Online. Visual Servoing Platform. Dostupné z: https://visp-doc.inria.fr/doxygen/visp-daily/classvpAdaptiveGain.html. [cit. 2025-02-15].
[48] LIU, Yang; XIE, Zongwu; GU, Yikun; FAN, Chunguang; ZHAO, Xiaoyu et al., 2017. Trajectory Planning of Robot Manipulators Based on Unit Quaternion. Online. In: *2017 IEEE International Conference on Advanced Intelligent Mechatronics (AIM)*. Munich, Germany: IEEE. Dostupné z: https://doi.org/10.1109/AIM.2017.8014189. [cit. 2025-03-22].
[49] THE MATHWORKS, INC., 2024. *What Is Camera Calibration?* Online. MathWorks. Dostupné z: https://www.mathworks.com/help/vision/ug/camera-calibration.html. [cit. 2025-02-15].
[50] SORIANO-SALVADOR, Enrique; MARTÍNN-RICO, Francisco a MÚZQUIZ, Gorka Guardiola, 2024. Implementing a Robot Intrusion Prevention System (RIPS) for ROS 2. Online. *ArXiv*. Dostupné z: https://doi.org/10.48550/arXiv.2412.19272. [cit. 2025-03-30].
[51] MACENSKI, Steve; SORAGNA, Alberto; CARROLL, Michael a GE, Zhenpeng, 2023. Impact of ROS 2 Node Composition in Robotic Systems. Online. *ArXiv*. Dostupné z: https://doi.org/10.48550/arXiv.2305.09933. [cit. 2025-02-15].
[52] INRIA, 2025. *Markerless generic model-based tracking using a RGB-D camera*. Online. Visual Servoing Platform. Dostupné z: https://visp-doc.inria.fr/doxygen/visp-daily/tutorial-tracking-mb-generic-rgbd.html. [cit. 2025-03-20].
[53] COMPORT, Andrew; MARCHAND, Eric; PRESSIGOUT, Muriel a CHAUMETTE, François, 2006. Real-time markerless tracking for augmented reality: the virtual visual servoing framework. Online. *IEEE Transactions on Visualization and Computer Graphics*. Roč. 12, č. 4, s. 615-628. Dostupné z: https://inria.hal.science/inria-00161250v1/document. [cit. 2025-03-20].
[54] TRINH, Souriya; SPINDLER, Fabien; MARCHAND, Eric a CHAUMETTE, François, 2018. A modular framework for model-based visual tracking using edge, texture and depth features. Online. In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Madrid, Spain, s. 89-96. Dostupné z: https://doi.org/10.1109/IROS.2018.8594003. [cit. 2025-03-21].
[55] INRIA, 2025. *Markerless generic model-based tracking using a color camera*. Online. Visual Servoing Platform. Dostupné z: https://visp-doc.inria.fr/doxygen/visp-daily/tutorial-tracking-mb-generic.html. [cit. 2025-03-20].

[56] BMR TEAM, 2025. *Brno Mars Rover*. Online. Dostupné z: https://brnomarsrover.cz/. [cit. 2025-04-15].
[57] EUROPEAN SPACE FOUNDATION, 2025. *European Rover Challenge.* Online. Dostupné z: https://roverchallenge.eu. [cit. 2025-04-15].
[58] HOLBA, Jan, 2025. *Master Thesis – Robotic Manipulator With a Visual Feedback*. Online. In: GOOGLE. YouTube. Dostupné z: https://www.youtube.com/watch?v=xMTTZYzh8YM. [cit. 2025-04-23].

SEZNAM PŘÍLOH

Příloha A - Přiložené USB

1. Zdrojový kód programu
2. Videodokumentace
3. Elektronická verze textu diplomové práce