ZÁPADOČESKÁ UNIVERZITA V PLZNI

Fakulta elektrotechnická
Katedra elektrotechniky a počítačového modelování

DIPLOMOVÁ PRÁCE

Návrh a implementace nového API robota DOBOT MG400

Autor práce: **Bc. Filip Hrádek**
Vedoucí práce: **Ing. Petr Kropík, Ph.D.**

2023

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta elektrotechnická
Akademický rok: 2022/2023

ZADÁNÍ DIPLOMOVÉ PRÁCE
(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Filip HRÁDEK**
Osobní číslo:	**E21N0037P**
Studijní program:	**N0714A060013 Elektronika a informační technologie**
Specializace:	**Elektronika**
Téma práce:	**Návrh a implementace nového API robota DOBOT MG400**
Zadávající katedra:	**Katedra elektroniky a informačních technologií**

Zásady pro vypracování

1. Analyzujte současné API robotického manipulátoru výrobce DOBOT (výuková řada).
2. Analyzujte potřebné klíčové prvky API a jeho návrhu pro řadu DOBOT MG400.
3. Implementujte API v Pythonu s důrazem na snadnost a přehlednost jeho použití a přehlednost.
4. Otestujte vytvořené API dle požadované metodiky –posun na pozici, směr, atd.
5. Vytvořte dokumentaci navrženého a otestovaného API.

Rozsah diplomové práce: **40 – 60**
Rozsah grafických prací:
Forma zpracování diplomové práce: **elektronická**

Seznam doporučené literatury:

1. Kinematika robotů [online]. (c) 2015 [cit. 13.4.2022].
   Dostupné z: http://cw.felk.cvut.cz/wiki/_media/courses/a3b99ro/robotismutnycz.pdf
2. Encyklopedie fyziky – kinematika [online]. (c) 2019 [cit. 13.4.2022].
   Dostupné z: http://fyzika.jreichl.com/main.article/view/4-kinematika
3. Seriál Programovací jazyk Lua [online]. (c) 2022 [cit. 13.4.2022].
   Dostupné z: https://www.root.cz/serialy/programovaci-jazyk-lua
4. Python v ČR [online]. (c) 2022 [cit. 13.4.2022]. Dostupné z: https://python.cz
5. python.org [online]. (c) 2022 [cit. 13.4.2022]. Dostupné z: https://www.python.org

Vedoucí diplomové práce: **Ing. Petr Kropík, Ph.D.**
Katedra elektrotechniky a počítačového modelování

Datum zadání diplomové práce: **7. října 2022**
Termín odevzdání diplomové práce: **26. května 2023**

L.S.

Prof. Ing. Zdeněk Peroutka, Ph.D.
děkan

Doc. Ing. Jiří Hammerbauer, Ph.D.
vedoucí katedry

V Plzni dne 7. října 2022

Abstrakt

Předkládaná diplomová práce je zaměřena na návrh a implementaci nového API robotických manipulátorů DOBOT MG400 a uArm Swift Pro v jazyce Python. Právě podpora v Pythonu je nezbytná pro potřeby současné testovací platformy. Ta slouží pro testování senzorů, jejichž funkce je potřeba otestovat jiným způsobem, než dovoluje v nich integrovaná diagnostika. Pro ověření jejich funkcionality je tak třeba zvolit odlišného přístupu a měnit fyzikální veličiny s nimi související. K tomu může sloužit právě robotický manipulátor, který pomůže nasimulovat požadované fyzikální vlastnosti. Pro možnost automatizace testování je klíčové mít jednoduché a zároveň robustní API, které je možné integrovat do již fungující testovací platformy.

Teoretická část práce je věnována teorii ohledně robotických manipulátorů z hlediska jejich konstrukce a komponentů, kinematiky, plánování pohybu a jejich řízení. Následuje nastínění problematiky testování senzorů pomocí robotického manipulátoru. Na závěr teoretické části je uveden přehled a parametry vybraných robotických manipulátorů.

Praktická část práce se zabývá analýzou stávajících API a SDK manipulátorů DOBOT MG400 a uArm Swift Pro. Na základě jejich klíčových prvků a metod, potřebných pro cílovou aplikaci, je navržen koncept nového API obsahující základní metody pro ovládání obou manipulátorů. Následně je vytvořené API otestováno dle požadované metodiky formou jednoduchého skriptu, který obsahuje testovací sekvenci čtečky čárových kódů. Poslední částí je tvorba dokumentace navrženého a otestovaného API.

Klíčová slova

Robotický manipulátor, testování senzorů, Python, návrh a implementace API, DOBOT MG400, uArm Swift Pro

Abstract

The presented master thesis focuses on the design and implementation of a new API for robotic manipulators DOBOT MG400 and uArm Swift Pro in Python. It is the Python support that is essential for the needs of the current testing platform. The latter is used for testing sensors whose functions need to be tested in a different way than the built-in diagnostics allow. Thus, in order to verify their functionality, a different approach has to be taken and the physical quantities associated with them have to be changed. A robotic manipulator can be used to help simulate the required physical properties. Having a simple yet robust API that can be integrated into an existing testing platform is key to being able to automate testing.

The theoretical part of the thesis is devoted to the theory of robotic manipulators in terms of their design and components, kinematics, motion planning and control. This is followed by an outline of sensor testing using a robotic manipulator. At the end of the theoretical part, an overview and parameters of selected robotic manipulators are presented.

The practical part of the thesis deals with the analysis of existing APIs and SDKs of DOBOT MG400 and uArm Swift Pro manipulators. Based on their key features and methods required for the target application, a new API concept containing basic methods for controlling both manipulators is proposed. Subsequently, the developed API is tested according to the required methodology in the form of a simple script that contains the test sequence of the barcode reader. The last part is the documentation of the designed and tested API.

Key Words

Robotic manipulator, sensor testing, Python, API design and implementation, DOBOT MG400, uArm Swift Pro

Poděkování

Tímto bych rád poděkoval vedoucímu diplomové práce Ing. Petru Kropíkovi, Ph.D. za poskytování cenných profesionálních rad a připomínek. Dále děkuji za kolegiální přistup a ochotu při vedení práce.

Obsah

Seznam symbolů a zkratek

x, y, z	Skalární souřadnice koncového efektoru (mm)
API	Application Programming Interface
CNC	Computer Numerical Control
DC	Direct Current
FEL	Fakulta elektrotechnická
HIL	Hardware In the Loop
HTML	HyperText Markup Language
I/O	Input/Output
LAN	Local Area Network
RAM	Random Access Memory
ROS	Robot Operating System
SCARA	Selective Compliance Assembly Robotic Arm
SDK	Software Development Kit
UART	Universal Asynchronous Receiver/Transmitter
ZČU	Západočeská univerzita
Φ	Úhel natočení koncového efektoru (°)

Úvod

Existuje celá řada senzorů, jejichž funkce je potřeba otestovat jiným způsobem, než dovoluje v nich integrovaná diagnostika. Může jít o indukční senzory, nebo různé světelné závory a podobně. Pro ověření jejich funkcionality je tak třeba zvolit odlišného přístupu a měnit fyzikální veličiny s nimi související. K tomu může sloužit například robotický manipulátor, který pomůže nasimulovat požadované fyzikální vlastnosti. Pro možnost automatizace testování je klíčové mít jednoduché a zároveň robustní API, které je možné integrovat do již fungující testovací platformy. Cílem této diplomové práce je návrh a implementace API pro robotické manipulátory DOBOT MG400 a uArm Swift Pro v jazyce Python. Právě podpora v Pythonu je nezbytná pro potřeby současné testovací platformy, která je založena na Robot Framework.

Pro řešení problému je potřeba provést analýzu současného API obou manipulátorů a zaměřit se přitom na všechny potřebné metody a klíčové prvky pro požadavky současné testovací platformy. Na základě této analýzy je ve finále možné sjednotit API obou manipulátorů a vytvořit nadstavbu ve formě wrapperu, který by umožnil, aby bylo jejich použití a ovládání co nejjednodušší a nejpřehlednější.

Teoretická část práce je věnována teorii ohledně robotických manipulátorů z hlediska jejich konstrukce a komponentů, kinematiky, plánování pohybu a jejich řízení. Následuje nastínění problematiky testování senzorů pomocí robotického manipulátoru. Dále je v rámci teoretické části uveden přehled a parametry vybraných robotických manipulátorů. Na závěr teoretické části je popsán výběr způsobu komunikace a vývojového prostředí.

Praktická část práce se zabývá analýzou stávajících API a SDK manipulátorů DOBOT MG400 a uArm Swift Pro. Na základě jejich klíčových prvků a metod, potřebných pro cílovou aplikaci, je navržen koncept nového API obsahující základní metody pro ovládání obou manipulátorů. Následně je vytvořené API otestováno dle požadované metodiky formou jednoduchého skriptu, který obsahuje testovací sekvenci čtečky čárových kódů. Poslední částí je tvorba dokumentace navrženého a otestovaného API.

1 Robotické manipulátory

Díky neustálým pokrokům v robotice, nižším nákladům a čím dál tím většímu rozšíření v průmyslu, kde mají zastoupení takřka ve všech průmyslových odvětví, jsou robotické manipulátory dostupnější a jejich použití už není jen otázkou velkovýroby, ale i menších provozů. Příkladem může být právě automatizace testování senzorů, kde k testování postačí jednotky robotických manipulátorů, a i tak dojde ke značnému usnadnění a zefektivnění procesu. [1][2]

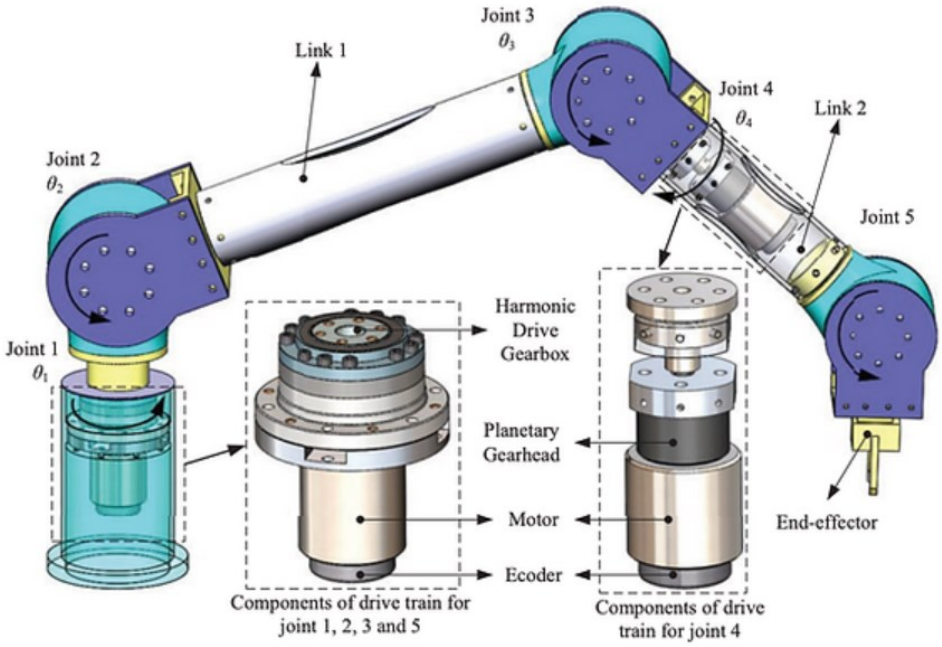
Robotické manipulátory jsou dnes již nedílnou součástí v moderním průmyslu a využívají se především na práci, která nemůže být jednoduše vykonána člověkem. Tím je myšleno to, že mohou vykonávat manuálně a energeticky náročné úkony s vysokým počtem opakování, a to s vysokou rychlostí a přesností. Umožňují tak automatizovat a optimalizovat výrobní proces. [1][2]

Z mechanického hlediska je robotický manipulátor tvořen kinematickým řetězcem tuhých těles (jednotlivá ramena), spojených klouby. Jeden konec řetězce je ukotven k základně, a na druhém konci je namontován koncový efektor. Výsledný pohyb konstrukce je získán složením elementárních pohybů každého ramene vzhledem k předchozímu. Moderní robotické manipulátory jsou schopny se pohybovat ve třech a více osách. To je dáno stupni volnosti, kde každý stupeň volnosti představuje možný rozsah pohybu v jednom směru. [1][5]

1.1 Konstrukce a komponenty robotického manipulátoru

Pro vysvětlení lze na začátek rozdělit typický robotický manipulátor na tři hlavní části:

- mechanická část (sada jednotlivých ramen spojených klouby s motory v každé ose);
- elektronická část (řídící elektronika a systém zodpovědný za plánování pohybů);
- počítačová část (vývojové prostředí, programovací jazyk, způsob programování). [1]



Obr. 1 Zjednodušená konstrukce robotického manipulátoru a jeho součástí [1]

1.1.1 Mechanické komponenty robotického manipulátoru

Jako celek se jedná o pohyblivou část manipulátoru, sloužící k provádění úkonů, díky níž je koncový efektor nastaven do požadované polohy. V následujících odstavcích bude uveden výčet a popis jednotlivých komponent, ze kterých se skládá tato část robotického manipulátoru. [1]

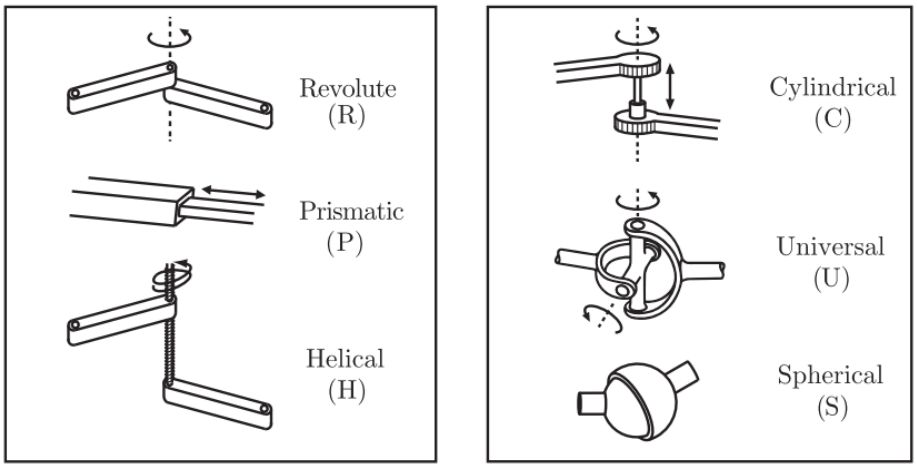
- **Rameno**

První ze zde vyjmenovaných mechanických částí manipulátoru jsou jednotlivá ramena, která propojují klouby manipulátoru. Jako materiál se nejčastěji používá hliník, který je levný a poskytuje dostatečnou pevnost vzhledem k jeho relativně nízké hmotnosti. To umožňuje snazší manipulaci, zvyšuje dynamiku a snižuje nároky na pohonné systémy manipulátoru. [5]

- **Kloub**

Kloub spojuje dvě ramena, čímž umožňuje jejich řízený nebo volný pohyb. Z toho plyne základní rozdělení na řízený nebo volně pohyblivý kloub. Řízený kloub obsahuje pohon, jehož poloha je ovládána řídícím systémem. Volně pohyblivý kloub na druhou stranu vlastní pohon neobsahuje, a jeho poloha je tak závislá čistě na poloze ostatních kloubů. Nejčastěji používanými typy řízených kloubů jsou posuvný (prizmatický) a rotační (otočný). Volný kloub lze sestrojit i jako sférický a válcový. Kvůli jednoduchosti mají klouby většinou jeden

stupeň volnosti, který poskytují mezi dvěma tělesy, která spojují. Tím zároveň odjímají zbylých pět stupňů volnosti jednoho prostorového tuhého tělesa vůči druhému. U běžných manipulátorů s otevřeným kinematickým řetězcem, kde je každý kloub ovládán svým motorem nezávisle na ostatních kloubech, je počet stupňů volnosti manipulátoru jednoduše roven počtu kloubů. To platí v případě, kdy má každý kloub právě jeden stupeň volnosti. U uzavřeného kinematického řetězce, který obsahuje i volně pohyblivé klouby, je výpočet složitější a počet stupňů volnosti manipulátoru nemusí být roven čistě počtu jeho kloubů. [3][5]



Obr. 2 Typy kloubů robotického manipulátoru [4]

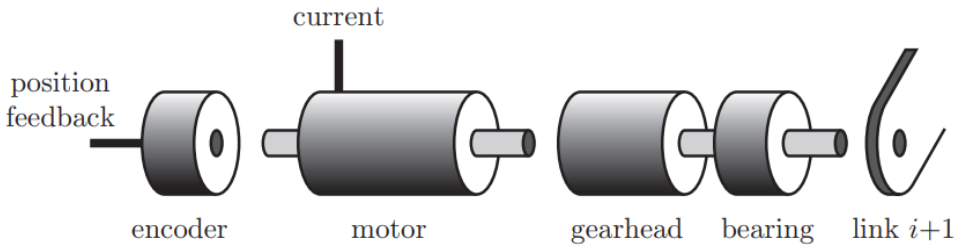
- **Pohon**

U menších manipulátorů jde obecně o to zajistit vysoce citlivé a přesné polohování, takže motor musí být schopen dosáhnout přesného úhlu natočení a řízení rychlosti. Proto jsou nejčastěji používaným typem elektrické pohony, jako krokové motory a servomotory. U větších manipulátorů, kde je potřeba vyšší nosnost, se používají i pneumatické a hydraulické pohony.

Příkladem zde může být právě servomotor, to je typ elektrického motoru, vhodný pro použití v menším robotickém manipulátoru, protože je uzpůsoben pro vysoce přesné aplikace a vysokorychlostní operace. Na druhou stranu je žádoucí, aby byl motor schopný operovat i při malých rychlostech otáčení pro pomalé a precizní pohyby. K tomu slouží převodovka, se kterou je spojen jeden konec hřídele motoru. Ta zajišťuje zvýšení točivého momentu za současného zmenšení rychlosti. Na druhém konci hřídele motoru je umístěn rotační enkodér, který snímá polohu rotoru a slouží tak jako zpětná vazba pro dosažení přesné regulace pohybu. [4]

- **Převodovka**

Převodovka je rozhraní mezi motorem, který generuje mechanickou energii pro vykonání daného pohybu, a koncovým efektorem, který výsledný pohyb vykonává. Používá se v případě, pokud není motor schopen vygenerovat dostatečný točivý moment tak, aby mohl být připojen přímo ke kloubu. Primárně tedy slouží pro snížení rychlosti otáčení a současně pro zvýšení točivého momentu, abychom splnili požadavky, kterých jen za pomocí servomotorů nedosáhneme. Na převodovce závisí i přesnost, a proto by měla mít velmi nízký prokluz a co nejmenší vůli. Nejčastěji se jako převody používají ozubená kola. [4][6]



Obr. 3 Grafické znázornění jednotlivých součástí kloubu [4]

1.1.2 Řídící elektronika robotického manipulátoru

Řídící jednotka je zodpovědná za řízení jednotlivých částí manipulátoru, jako je ovládání servomotorů, senzorů a měničů rychlosti. Řídící jednotka přijímá vstupy od uživatele například ve formě požadované trajektorie a zároveň bere v úvahu i zpětnou vazbu o poloze ze snímačů umístěných v každém kloubu. Posléze je na základě požadované trajektorie, modelu dynamiky manipulátoru a naměřené odchylce aktuálního stavu manipulátoru od požadovaného, vypočítán požadovaný točivý moment každého akčního členu. Dále lze řídící jednotku rozdělit na dvě hlavní části. [2][4]

- **Mozek manipulátoru**

Část řídící jednotky, která přijímá a zpracovává instrukce, na jejichž základě plánuje trajektorii dle zvoleného algoritmu. Cílem je sestavit matematický model popisující vstupně-výstupní vztahy charakterizující jednotlivé součásti manipulátoru. Toho je dosaženo na základě analýzy informací ohledně vlastností mechanické konstrukce, akčních členů a zpětné vazby ze senzorů. Díky modelu je možné nalézt vhodnou strategii řízení pohybu, a tak správně provádět požadované úlohy. Problematika modelování, plánování trajektorie a řízení manipulátorů bude probrána v následujících kapitolách. [1][5]

- **Ovladače motorů**

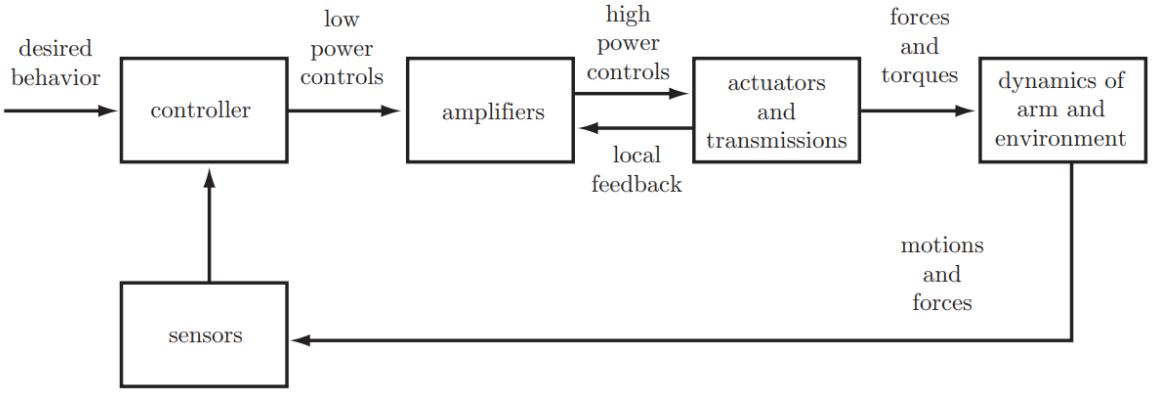
Elektronika v servomotoru pracuje s nízkým napětím, které však není dostatečné k jeho pohonu, proto je úkolem ovladače motoru zesílit nízkonapěťové signály na úroveň požadovanou pro řízení servomotoru. Kromě požadované velikosti je potřeba dle typu motoru dodávat i napětí určitého tvaru, ať jde o stejnosměrné napětí u DC servomotorů s permanentními magnety anebo střídavé napětí u bezkartáčových servomotorů. Dále zajišťují díky zpětné vazbě ze snímačů polohy motoru přesné řízení manipulátoru. Jako snímač se používá absolutní enkodér, který informaci o poloze dodává ovladači motoru nejen když je manipulátor zapnutý, ale pamatuje si polohu i v případě nečekaného výpadku napájení, kde musí servomotor znát polohu před opětovným zahájením práce, a tak zabraňuje chybám. [1][5]

- **Senzory**

Senzory poskytují zpětnou vazbu a informace ohledně stavu manipulátoru a jeho okolí. V základu lze rozdělit senzory na interní a externí.

Interní senzory poskytují informace o poloze manipulátoru, jeho rychlosti, zrychlení a točivém momentu. Mezi základní senzory patří detektor kolize, který je založen na principu měření působící síly nebo snímání točivého momentu tak, aby manipulátor věděl, co se děje v daných osách. Na základě toho je manipulátor schopen detekovat objekty a překážky v jeho cestě a vyvarovat se tak kolizím. Pro měření rychlosti lze použít tachometry a jako snímače polohy motorů se používají enkodéry pro poskytnutí zpětné vazby řídící jednotce. Existují inkrementální enkodéry, kde musí stroj při každém zapnutí znovu najíždět na kalibrační čidlo, aby souhlasilo odměřování polohy, a absolutní enkodéry, které jsou vhodnější, protože si pamatují svou polohu i po vypnutí stroje. [4][6]

Externí senzory poskytují informace o okolí manipulátoru, aby byl obeznámen s uspořádáním prostředí, ve kterém pracuje. U některých aplikací je potřeba, aby manipulátor snímal své okolí a detekoval objekty, se kterými chce manipulovat. K tomu se často využívají vizuální snímače (kamery). Dále existují i akustické senzory a senzory vzdálenosti a přiblížení (laserový dálkoměr). Data ze senzorů jsou posílána zpět do řídící jednotky, kde je na jejich základě naplánován pohyb koncového efektoru. [4][6]



Obr. 4 Řídicí systém robotického manipulátoru [4]

- **Koncový efektor**

Část manipulátoru, která slouží k interakci s okolním prostředím nebo manipulovaným objektem. Většinou lze koncový efektor jednoduše vyměnit a přizpůsobit tak manipulátor různým aplikacím a být multifunkčním. Dle zamýšlené aplikace může být koncový efektor ve například formě vakuové přísavky, gripperu, 3D tiskové hlavy, elektromagnetu, vrtačky nebo svářečky. Při polohování se udává pozice koncového efektoru jako relativní vůči pozici základny manipulátoru. [3]

1.1.3 Způsoby programování robotického manipulátoru

K tomu, aby bylo umožněno operátorovi ovládat manipulátor pomocí zadávání pokynů u úlohám, které má manipulátor provést a dále ho jednoduše programovat, je zapotřebí uživatelské rozhraní mezi uživatelem a daným manipulátorem, které je podporované vhodnými programovacími jazyky. Toto rozhraní slouží kromě překladu příkazů ve specifickém programovacím jazyce ke kontrole správného provedení úloh, které manipulátor vykonává tak, aby byla manipulace co nejsnadnější a nejbezpečnější. Jde o to zajistit například snadné popsání pohybů manipulátoru v prostoru nebo automaticky vykovávat akce založené na zpětné vazbě ze senzorů. Podle cílové aplikace je zvolen vhodný způsob programování manipulátoru. [2][5]

- **Učení pomocí předvádění**

Nejstarší metodou je učení pomocí předvádění (drag-n-teach). Jde o to, že uživatel učí manipulátor novým pohybům k provedení určité úlohy tak, že jej fyzicky posouvá do požadovaných poloh, ve kterých si manipulátor ukládá kartézské souřadnice. Souřadnice pak určují posloupnost průjezdových a pracovních poloh manipulátoru v daných časech. Na základě těchto vstupních hodnot následně algoritmus pro generování trajektorie

naplánuje výsledný pohyb, kde jsou specifikovány rychlostní profily kloubů a doba trvání pohybu. U menších manipulátorů lze zadávat pohyby pomocí ručního navádění. U větších průmyslových manipulátorů se využívá speciálního zařízení (pendant), pomocí kterého lze navádět manipulátor a učit ho jednotlivé kroky. [2][6]

- **Programovací jazyky na úrovni úloh**

Cílem je umožnit přímé řízení pohybů a akcí manipulátoru pomocí vysokoúrovňových jazyků. Může jít o programovací prostředí a jazyk určený specificky pro robotické manipulátory, nebo se používají již existující strukturované vysokoúrovňové programovací jazyky s použitím speciálních knihoven a API s potřebnými funkcemi, které jsou vyžadovány robotickými aplikacemi. Tuto metodu lze nazvat i objektově orientovaným programováním, protože lze zadávat komplexní úlohu složenou z více kroků pomocí vysokoúrovňových příkazů (objektů). Zadáváním příkazů pouze k požadovaným dílčím cílům úlohy docílím automatického provedení řady akcí, aniž bych musel specifikovat detaily každé akce, kterou má robot provést. I u manipulátorů, které jsou uzpůsobeny této metodě programování, zůstává možnost programování pomocí učení. [5][6]

- **Off-line programování**

Tato metoda umožňuje programovat manipulátor, aniž by byl fyzicky dostupný pro programátora, za pomoci simulace trajektorií. K tomu je potřeba programovací prostředí rozšířené o grafické rozhraní, kde je znázorněn model manipulátoru. Programy lze vytvářet a testovat bez přístupu k samotnému manipulátoru a následně je po ověření načíst do manipulátoru. Výhodou je, že v případě potřeby přeprogramování nemusí být manipulátor odstaven. Příkladem softwaru založeném na tomto způsobu programování může být Coppelia robotics. [2]

1.2 Kinematika, plánování pohybu a řízení robotického manipulátoru

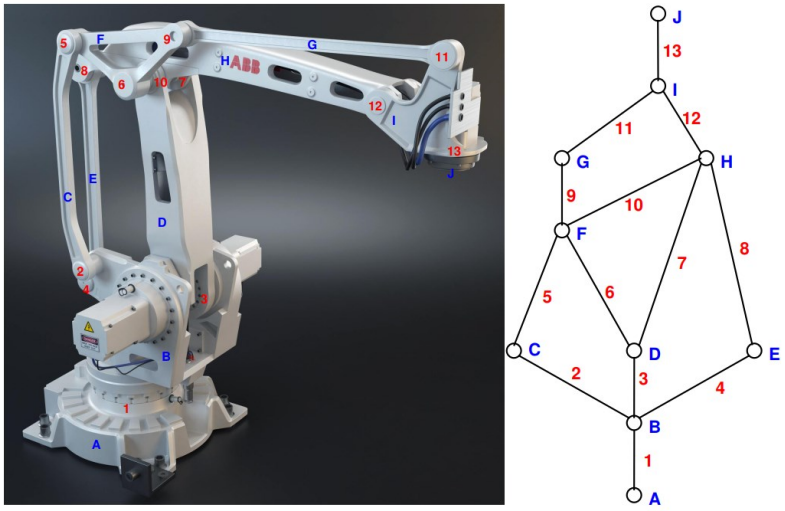
1.2.1 Kinematika

Kinematika je věda zabývající se pohybem bez ohledu na síly, které ho způsobují. Studium kinematiky manipulátorů se tedy týká všech geometrických a časových vlastností pohybu a zabývá se polohou, rychlostí a zrychlením. [6]

Využíváme matematický popis mechanismu, pomocí kterého jsme schopni pro každý časový okamžik získat polohu, rychlost a zrychlení libovolného bodu. Příkladem může být nutnost popsat polohu a orientaci koncového efektoru, aby bylo možné manipulovat

s objektem v prostoru. Manipulátor schematicky znázorňujeme jako kinematický řetězec tuhých těles (ramen) spojených pomocí kloubů, které umožňují relativní pohyb sousedních ramen. Mezi základní způsoby matematického popisu mechanismu patří trigonometrické vyjádření, maticový způsob a vektorová metoda. [5][6]

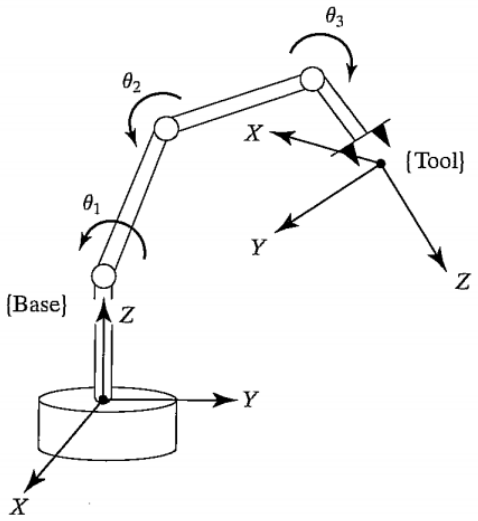
Při realizaci modelu mechanismu můžeme narazit na pojmy jako kinematická dvojice, což je dvojice ramen spojených kloubem. Mezi druhy kinematických dvojic patří stejně jako u jednotlivých typů kloubů rotační, posuvná, válcová, sférická a plochá. Kinematický řetězec jakožto množina ramen spojených klouby je reprezentován grafem, kde uzly grafu představují ramena a hrany představují klouby. Kinematický řetězec má jedno rameno připevněno k základně a na druhém konci řetězce se nachází koncový efektor. Kinematické řetězce se nejčastěji dělí na otevřené, uzavřené a smíšené. U otevřeného řetězce jsou ramena propojena v sérii a čistě spojují začátek a konec řetězce, podmínkou je, že všechny klouby musí být řízené. U uzavřeného řetězce se nachází i volně pohyblivé klouby, a tak tvoří určitá sekvence ramen smyčku. [3][5]



Obr. 5 Příklad grafu smíšeného kinematického řetězce [3]

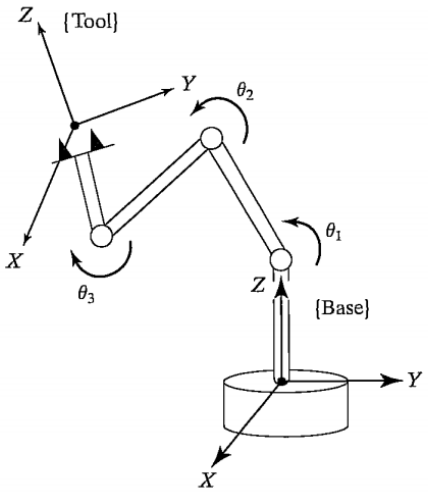
Z pohledu kinematiky existují dva základní algoritmy při řešení mechanické manipulace. Dopředná a inverzní kinematická úloha.

Při řešení dopředné kinematiky jde o statický geometrický problém výpočtu aktuální polohy a orientace koncového efektoru. Ta je odvozená od konfigurace kloubů neboli pozice (úhlů natočení) a případně na dalších vstupních veličinách jako délka ramen. Poloha a orientace koncového efektoru, která je dána pozicí jednotlivých kloubů se zjišťuje vůči nulovým souřadnicím, což je poloha základny manipulátoru. Při řešení úlohy dopředné kinematiky existuje vždy právě jedno řešení. To je dáno tím, že je poloha a orientace koncového efektoru pro každé uspořádání kloubů jednoznačná. [5][6]



Obr. 6 Pozice koncového efektoru vzhledem k pozici základny manipulátoru je funkcí konfigurace jednotlivých kloubů [6]

Při řešení inverzní kinematiky jde o opačný problém a je zde potřeba určit polohy (úhly natočení) kloubů tak, aby bylo dosaženo požadované konfigurace koncového efektoru. Inverzní kinematika je obecně složitější než dopředná vzhledem k tomu, že pro dosažení konkrétní polohy a orientace koncového efektoru může existovat více řešení uspořádání poloh kloubů nebo nemusí existovat žádné řešení. To, jestli existuje řešení pro dané zadání či nikoli, definuje pracovní prostor manipulátoru. [5][6]



Obr. 7 Pro danou polohu a orientaci koncového efektoru je dopočítána poloha jednotlivých kloubů [6]

1.2.2 Dynamika

V souvislosti s řízením polohy manipulátoru je potřeba se zabývat i dynamikou, kde řešíme pohyb s ohledem na síly a momenty, které ho způsobují. Analogicky ke kinematice existují pojmy jako dopředná a inverzní dynamika. V úloze dopředné

dynamiky potřebujeme určit výsledné zrychlení koncového efektoru pro dané síly a momenty jednotlivých kloubů. U inverzní dynamiky je opět opačná situace, kde potřebujeme určit vstupní síly a momenty kloubů pro dosažení požadovaného zrychlení koncového efektoru. Prostorové a časové parametry trajektorie, kterou má urazit koncový efektor, udávají přesnou podobu požadovaných funkcí momentů jednotlivých kloubů. Dále však záleží i na zatížení manipulátoru a faktorech, jako je tření v kloubech. [5][6]

1.2.3 Generování trajektorie

Jak již bylo zmíněno v kapitole ohledně řídící elektroniky manipulátoru, je úkolem řídící jednotky dopočítávat a automaticky generovat trajektorii pohybů na základě vstupních dat tak, aby nebylo potřeba specifikovat celou časovou historii každého kloubu nebo koncového efektoru. Trajektorie je složena z cesty v podobě posloupnosti bodů, kterých má manipulátor dosáhnout. Jedná se o čistě geometrický popis, kde je daná posloupnost hodnot dosažena za použití interpolační funkce (obvykle proložením polynomem daného řádu). Dále je trajektorie složena z časové posloupnosti, která zase určuje časy, kdy má být jednotlivých bodů dosaženo. Vstupní data tedy musí obsahovat stručný popis požadovaného pohybu ve formě uspořádané množiny průchozích bodů a s tím korespondující množině kontrolních časů. Algoritmus pro generování trajektorie je na základě těchto dat schopen vytvořit trajektorii pro každý kloub, která bude splňovat podmínky zadané uživatelem. Mezi nejčastější pohyby patří například pohyb z bodu do bodu (point-to-point), kde se koncový efektor pohybuje pouze po přímkách. Toho se využívá u jednoduchých úloh manipulace s objekty a jako vstupní data postačí jen pozice vyzvednutí a uvolnění objektu. Dále existují hladké trajektorie (spline) neboli dráhový pohyb, kde se pohybujeme po zadaných a načasovaných bodech. Toho se využívá například při obrábění, kde musí koncový efektor přesně sledovat požadovanou trajektorii. Posledním příkladem jsou časově optimalizované trajektorie po zadaných drahách s ohledem na dynamiku manipulátoru a limity akčních členů. [4][5]

1.2.4 Plánování pohybů

V určitých aplikacích se v pracovním prostoru manipulátoru můžou nacházet překážky nebo další manipulátory a je tak nutné se předem zorientovat v pracovním prostoru a naplánovat pohyb, aby nedocházelo ke kolizím. Systém musí mít k dispozici model manipulátoru, pracovního prostoru, možných překážek a dále se vyhnout omezením kloubů, akčních členů a dalším fyzikálním omezením. Obecně je potřeba nalézt bezkolizní cestu bez

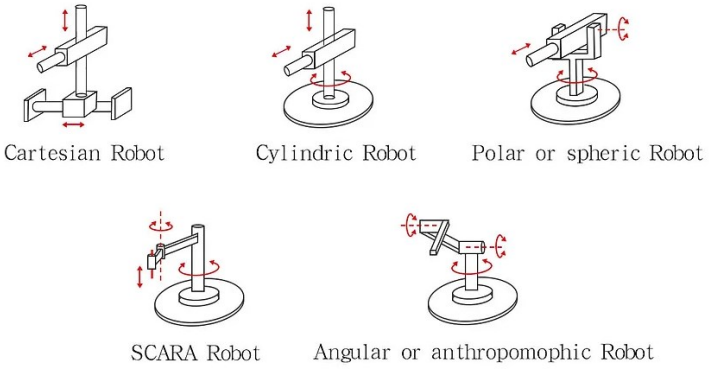
ohledu na dynamiku a dobu trvání pohybu. Nelze se zabývat jen problematikou plánování bezkolizní cesty koncového efektoru, protože ta spadá pod problematiku obecného plánování pohybu, kde je zapotřebí vzít v úvahu celou konstrukci manipulátoru tak, aby nenastala kolize žádné jeho části. V tomto případě se udává pojem kolizní prostor, což je prostor, který vyplní konstrukce manipulátoru v rámci své činnosti využitím všech možností svého pohybového systému. [4][5]

1.2.5 Řízení robotického manipulátoru

V závislosti na prováděné úloze a prostředí manipulátoru je potřeba měnit způsob, jakým je řízeno jeho chování. O to se opět stará řídící jednotka, která převádí zadání úlohy na časové průběhy sil a momentů akčních členů. Problematika řízení je složitá a syntézu těchto sil a momentů nelze provést pouze na základě dynamického modelu, protože zcela nepostihuje skutečnou konstrukci a ostatní proměnné, jako ovlivňování se jednotlivých kloubů mezi sebou svým pohybem. Aby byl zpětnovazební řídicí systém schopen splnit požadavky na přesnost provádění předepsané úlohy, je při řízení potřeba zohlednit informace ze senzorů, jako jsou síly vyměňované při kontaktu s prostředím nebo polohu objektů zjištěnou pomocí kamery. Pomocí této zpětnovazební smyčky je vypočítána odchylka mezi referenčními vstupy a údaji ze senzorů. Mezi základní strategie řízení patří řízení pohybu (polohy), silové řízení, impedanční řízení a hybridní řízení pohybu a síly. Silové řízení je založeno na schopnosti řízení kontaktních sil při dotyku a manipulaci s díly, nástroji a pracovními plochami. Řízení polohy má pak smysl použít, pokud se manipulátor pohybuje ve volném prostoru a není tak žádný povrch, na který by bylo potřeba reagovat. V případě použití pouze řízení polohy by mohla nastat situace, kde by se manipulátor dotýkal tuhého povrchu a v místě kontaktu by vznikla nadměrná síla, nebo že se kontakt s povrchem ztratí, i když byl žádoucí. Na druhou stranu pro silové řízení platí, že manipulátor není většinou omezen reakčními plochami ve všech směrech současně a je tak třeba zavést hybridní řízení polohy a síly současně. Silové řízení je však komplementární k řízení polohy a v jednom směru je většinou použitelná jen jedna varianta. U hybridního řízení jsou tak některé směry řízeny pomocí zákonu řízení polohy a zbylé směry zákonem řízení síly. [4][6]

1.3 Typy robotických manipulátorů

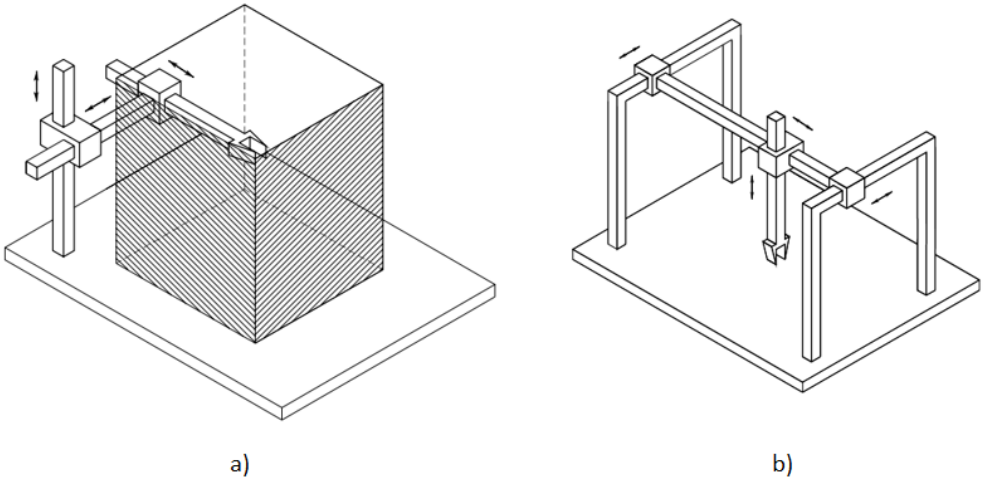
Existuje celá řada robotických manipulátorů, které se liší svou konstrukcí a aplikací. Rozdíly spočívají v tom, jak provádí dané pohyby, rozsahu pohybů a maximální nosnosti. Manipulátory můžeme do jednotlivých struktur klasifikovat na základě jejich prvních třech kloubů (os). U manipulátorů mají většinou první tři klouby (počítáno od rámu) velký rozsah pohybu a určují tak tvar a vlastnosti pracovní obálky. Zbylé klouby bývají nejčastěji otočné a zajišťují orientaci manipulovaného objektu. Ve zkratce jde o to, v jakém pořadí jsou jednotlivé klouby uspořádány a jaké mají stupně volnosti. V následujících odstavcích bude uveden popis nejpoužívanějších typů robotických manipulátorů. [1][3]



Obr. 8 Typy robotických manipulátorů [1]

1.3.1 Kartézské robotické manipulátory

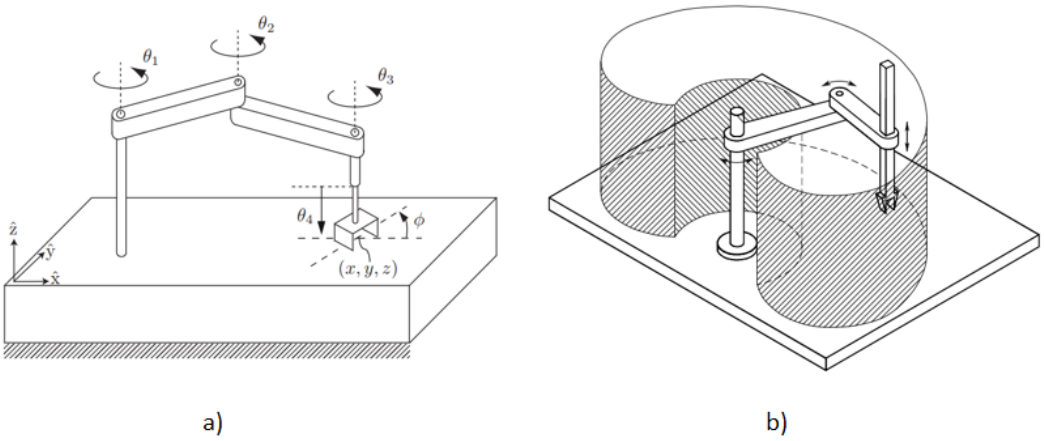
Také nazývány lineárními vzhledem k tomu, že je s objektem manipulováno v sekvenčních lineárních pohybech ve třech ortogonálních osách, kde je mechanismus činnosti manipulátoru založen na kartézském souřadném systému (x, y, z). Manipulátor je tvořen třemi lineárními (posuvnými) klouby, kde každý z nich slouží pro pohyb v dané ose. Tyto manipulátory jsou velice přesné, ale na druhou stranu méně obratné vzhledem k tomu, že jsou všechny klouby lineární. Koncové rameno může být dle aplikace umístěno vodorovně anebo svisle. Kartézský manipulátor může být i ve formě portálové struktury, která může mít velký pracovní prostor a manipulovat s objekty o velké hmotnosti. [1][5]



Obr. 9 Kartézský robotický manipulátor: a) pracovní prostor manipulátoru, b) portálová struktura [5]

1.3.2 SCARA robotické manipulátory

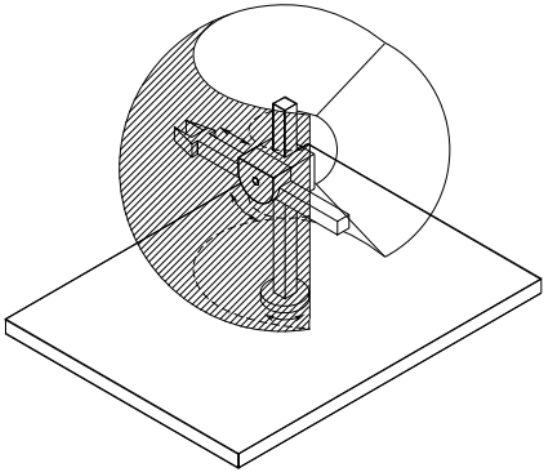
SCARA neboli Selective Compliance Assembly Robotic Arm. Struktura manipulátoru je většinou taková, že obsahuje první dva nebo tři paralelní rotační klouby pro pohyb a orientaci v rovině a jeden kloub je posuvný tak, aby byl zajištěn pohyb koncového efektoru svisle k rovině. Ve výsledku je koncový efektor schopný se pohybovat ve všech osách (x, y, z) a dále je jeho poloha popsána úhlem Φ, který udává orientaci v rovině x-y. Tyto manipulátory jsou vhodné pro vertikální operace nad rovinou, kde mají značný rozsah, proto se nejčastěji využívají na úlohy typu pick-and-place. Dále jsou zpravidla velmi rychlé, protože otočné klouby nepracují proti gravitaci a nemusí tak nést žádnou váhu, ať už manipulátoru nebo manipulovaného objektu. I přes to, že jsou rychlejší než kartézské manipulátory, je toto vykoupeno menší přesností. [3][4]



Obr. 10 SCARA robotický manipulátor: a) rozsah pohybů [4], b) pracovní prostor manipulátoru [5]

1.3.3 Sférické robotické manipulátory

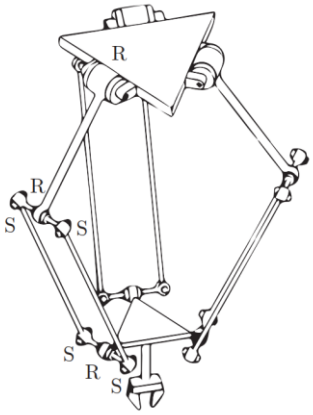
Sférické nebo také polární manipulátory patří mezi nejstarší technologii robotických manipulátorů. Manipulátor se pohybuje v polárních souřadnicích a výsledkem je tak sférická pohybová sekvence. V tomto případě má manipulátor tři stupně volnosti. Je tvořen rotující základnou a ramena jsou spojena lineárním a rotačním kloubem. [1][5]



Obr. 11 Sférický robotický manipulátor a jeho pracovní prostor [5]

1.3.4 Delta robotické manipulátory

Zde je uveden příklad na konstrukci složenou ze dvou plošin, kde je horní stacionární a dolní pohyblivá. Plošiny jsou spojeny třemi nohami, kde je každá z nich tvořena pěti rameny spojených třemi otočnými a čtyřmi sférickými klouby. Motory jsou umístěny v těle manipulátoru. Manipulátor má tři stupně volnosti, vzhledem k tomu, že je pohyblivá plošina s koncovým efektorem, který je nejčastěji ve formě vakuové přísavky, schopna se pohybovat v rovinách x-y-z. [1][4]

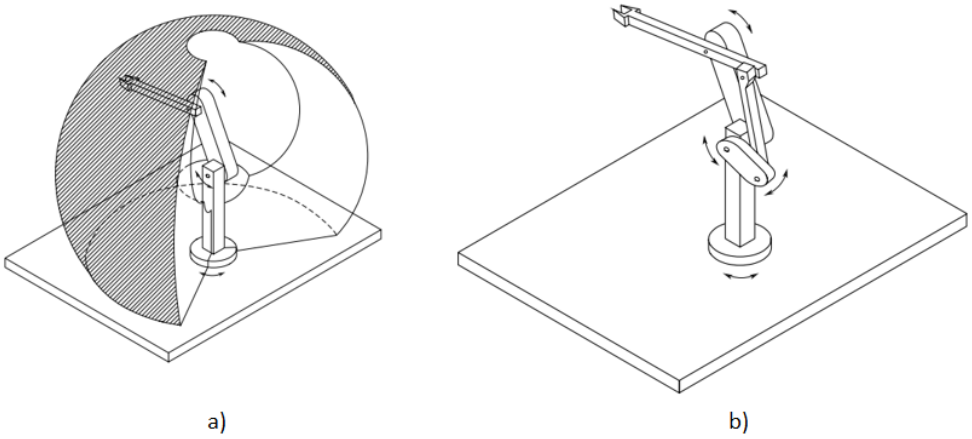


Obr. 12 Struktura delta robotického manipulátoru [4]

1.3.5 Angulární robotické manipulátory

Angulární manipulátory se svou konstrukcí podobají nejvíce lidské ruce a dokážou tak vyprodukovat velkou škálu pohybů. Manipulátor je nejčastěji tvořen rotačními klouby, kde v tomto případě osa spodního kloubu ortogonální vůči zbylým kloubům, které jsou pak na sebe kolmé. Pokud má manipulátor umožnit libovolnou polohu a orientaci manipulovaného objektu, je potřeba nejméně šest stupňů volnosti. Tyto manipulátory jsou rozšířené jako kolaborativní roboti, kteří většinou operují právě v šesti osách. Je však nutné brát v úvahu, že s větší volností narůstá obtíž s udržením dostatečné přesnosti, kde se více pohybů promítá do více polohových chyb a dále se manipulátor prodražuje a snižuje se jeho tuhost. [1][3]

Jak bylo zmíněno v předešlých kapitolách, manipulátor může být tvořen otevřeným kinematickým řetězcem, kde je každý kloub opatřen svým pohonem. Další možností je uzavřený kinematický řetězec, kde mezi sebou tvoří určitá ramena smyčku a vyskytují se zde i volně pohyblivé klouby. Problémem tohoto řešení je omezený počet stupňů volnosti a tím i pracovního prostoru, na druhou stranu je možné využít strukturu s paralelogramem pro dosažení vyšší nosnosti. [5]



Obr. 13 Angulární robotický manipulátor: a) pracovní prostor, b) struktura s paralelogramem [5]

1.4 Kritéria pro výběr robotického manipulátoru

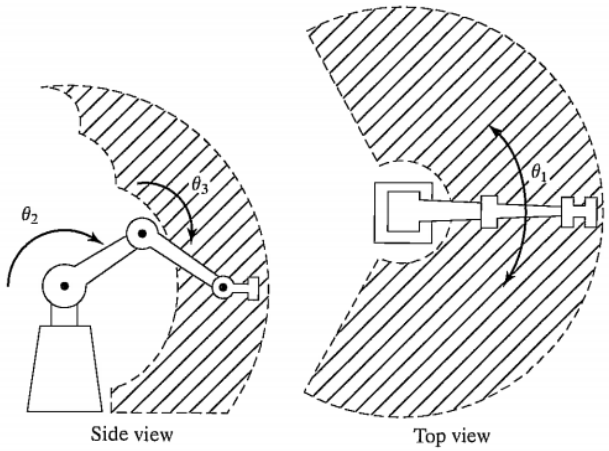
Existuje několik hlavních faktorů, podle kterých se při výběru robotického manipulátoru pro danou aplikaci rozhodovat.

1.4.1 Rozsah pohybů

Obecně se udává pracovní prostor manipulátoru a je to část prostoru, které může manipulátor se svým koncovým efektorem dosáhnout. Ve skutečnosti lze pracovní prostor

rozdělit na oblast, ve které je manipulátor schopen svým koncovým efektorem dosáhnout všech jeho možných orientací. Ta spadá pod větší oblast nazývanou dosažitelný pracovní prostor, který je ohraničen vzdáleností, které koncový efektor dosáhne alespoň s jednou jeho orientací. Maximální rozsah manipulátoru záleží na jeho topologii, velikosti a dále například na mechanických omezeních kloubů. [1][5]

Pokud je zapotřebí zvětšit pracovní prostor manipulátoru, lze použít zařízení pro zvýšení jeho dosahu, jako je například lineární pojezd nebo křížový stůl. [3]



Obr. 14 Pracovní prostor robotického manipulátoru [6]

1.4.2 Opakovatelnost

Zkráceně je to míra schopnosti manipulátoru vrátit se do dříve dosažené polohy. Opakovatelnost udává, jaký je rozptyl nebo korelace v polohách koncového efektoru, pokud manipulátor vícekrát provádí přesun na stejnou pozici. [1][5]

1.4.3 Přesnost

Přesnost závisí na konstrukci a rozměrech manipulátoru, kde platí, že s větším počtem stupňů volnosti narůstá potíž s udržením požadované přesnosti. Jde o rozdíl mezi požadovanou polohou, která je vypočtena pomocí dopředné kinematiky, a skutečně dosaženou polohou, kde vstupují do úvahy i další proměnné, jako mechanické tolerance. Přesnost se dále mění i v závislosti na poloze koncového efektoru v pracovním prostoru, kde s narůstající vzdáleností od základny manipulátoru klesá přesnost. [5]

1.4.4 Nosnost

Nosnost neboli užitečné zatížení se udává jako maximální hmotnost předmětu, se kterým je manipulátor schopen manipulovat. Stejně jako u přesnosti zde záleží na poloze

v pracovním prostoru a manipulátor nemusí být schopen s maximálním zatížením dosahovat maximálního rozsahu pohybů. [1]

1.4.5 Rychlost

V určitých aplikacích může být z ekonomického hlediska, kde chceme dosáhnout co největšího objemu výroby, žádoucí co nejvyšší rychlost. Naopak u aplikací, jako svařování či lakování stříkáním není potřeba manipulátor s co nejvyšší rychlostí, protože rychlost pohybu je ovlivněna hlavně samotným procesem. V souvislosti s maximální rychlostí je potřeba řešit i zrychlení manipulátoru. To lze uvést na příkladu úlohy pick-and-place, kde v manipulátor v rámci úlohy musí zrychlovat a poté zpomalovat v koncových bodech, kde je manipulováno s předmětem, tak aby bylo dosaženo určité opakovatelnosti a přesnosti. Udává se celková doba jednoho cyklu pro konkrétní úlohu, kde právě zpomalení a zrychlení zabírá většinu doby tohoto cyklu. [1][6]

1.4.6 Cena

Cena se odvíjí od všech zde vyjmenovaných parametrů robotických manipulátorů. Rozhoduje tak cílová aplikace manipulátoru, kde na příkladu testování senzorů nejsou kladeny vysoké nároky na opakovatelnost a užitečné zatížení. Proto lze u těchto nenáročných aplikací nahradit drahé průmyslové manipulátory cenově dostupnějšími stolními manipulátory.

2 Testování senzorů pomocí robotického manipulátoru

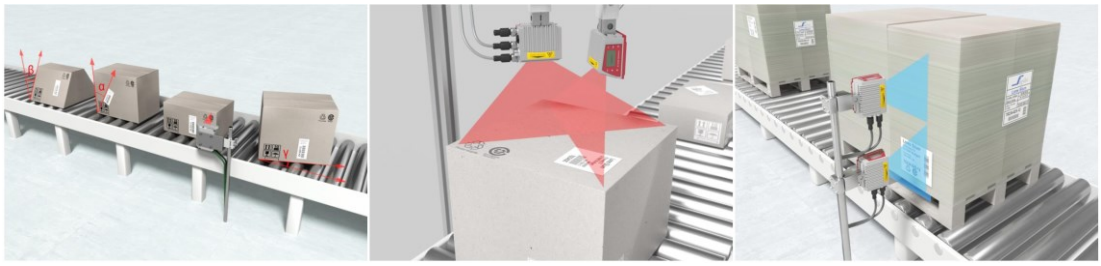
Před výběrem vhodného robotického manipulátoru je třeba se zaměřit na to, jaká úskalí přináší zamýšlená aplikace a lépe díky tomu pochopit, jaké požadavky jsou na manipulátor, jeho řízení a programování kladeny. Dále je díky tomu možné se zaměřit na to, jakým způsobem navrhnout nové API, aby bylo dosaženo požadovaných výsledků.

Firma Leuze se zabývá integračním, systémovým a akceptačním testováním senzorů. Senzory vyvíjené firmou Leuze slouží pro nasazení ve výrobním procesu, kde například nahrazují manuální čtení čárových kódů za automatické a tím napomáhají ke zrychlení a zefektivnění výroby. Před nasazením senzorů do výroby musí však proběhnout jejich otestování pro ověření, zdali vyvinuté zařízení splňuje požadavky zákazníka. Zde je cílem usnadnit a zefektivnit proces testování celé řady senzorů, jejichž funkce je potřeba mnohdy otestovat jiným způsobem, než dovoluje v nich integrovaná diagnostika. Může jít o indukční senzory, různé světelné závory nebo čtečky čárových kódů. Tohoto požadavku lze dosáhnout pomocí implementace robotických manipulátorů, které pomůžou nasimulovat požadované fyzikální vlastnosti a můžeme tak pro ověření funkcionality senzorů zvolit odlišného přístupu a měnit fyzikální veličiny s nimi související. Na konci testovacího procesu musí být výsledkem zařízení, které čte automatizovaně čárové kódy, lze ho připojit do stávající průmyslové sítě a musí být schopno pracovat v daných podmínkách a bez obsluhy.

Obr. 15 Nasazení čtečky ve výrobě: a) čtečka čárových kódů, b) automatizace čtení čárových kódů

Při testování se používá metoda HIL (Hardware In the Loop), jakožto simulace reálného prostředí. Příkladem testovacího scénáře je požadavek, že čárový kód musí být přečten. Otestuji proto senzor čtením kódu tisíckrát po sobě, a test opakuji pro různé světelné podmínky. Dalším požadavkem je, že přečtený čárový kód musí být expedován na síť. Při tom loguji komunikaci a sleduji, že jsou všechny kódy správně expedovány na uměle vytvořenou síť.

I po finálním nasazení senzoru, kdy jsou splněny zákaznické požadavky a jsou otestované provozem, vývojový cyklus nekončí a zařízení je v tzv. v módu maintenance. Zde jsou případné zjištěné chyby opravovány a vychází nový software řešící specifické požadavky daného nasazení. Zjištěné nedostatky, a případně inovace, jsou reflektovány do nově vyvíjených zařízení a tato zpětná vazba může být kromě toho implementována i do způsobu testování, kde může například robotický manipulátor dobře simulovat situaci z výroby, kdy dochází k chybnému vyčítání z důvodu špatné orientace čárového kódu. Manipulátor tak může čtečku navíc testovat i přikládáním čárových kódů pod různými úhly jen za pomocí změny orientace koncového efektoru.



Obr. 16 Znázornění chybného vyčítání z důvodu špatné orientace čárového kódu

3 Přehled robotických manipulátorů vhodných pro testování

Na základě znalosti cílové aplikace lze zvolit vhodný manipulátor, který bude splňovat požadované vlastnosti. V úvahu připadají komerčně dostupné manipulátory, jako je DOBOT MG400 a uArm Swift Pro. V obou případech se jedná o stolní robotické manipulátory, které jsou převážně určené pro nenáročné aplikace, jako jsou například výukové účely a malosériová výroba, pod co spadá právě testování senzorů. Oba manipulátory zároveň spadají do kategorie kolaborativních robotů. To znamená, že jsou uzpůsobeni práci po boku lidí a není potřeba zvláštních bezpečnostních opatření, jako například použití bezpečnostních bariér. Oba manipulátory mohou být, díky možnosti namontování různých koncových efektorů, přizpůsobeni velké škále aplikací, ať už jde o vakuové přísavky, grippery, nebo dovybavení manipulátoru o vizuální systém. Oba manipulátory jsou koncipované jako open-source platforma a výrobci poskytují řadu zdrojů pro podporu vývojářů, tak aby bylo možné pro manipulátory vytvářet své vlastní aplikace.

V následujících kapitolách budou představeny právě tyto manipulátory z hlediska jejich konstrukce, parametrů, možností ovládání a programování. Cílem je na základě těchto informací vytvořit společné API pro oba manipulátory.

Vzhledem k cenové dostupnosti DOBOT MG400 se již nejedná o manipulátor vhodný čistě pro výukové účely, a proto se hledala vhodná a dostupnější alternativa s podobnými parametry. Firma DOBOT takovou alternativu nabízí v podobě DOBOT Magician, což je odlehčená verze, vyvinuta primárně pro výukové účely. DOBOT Magician je však svými parametry velmi podobný manipulátoru uArm Swift Pro a byl tak jako možnost nahrazen právě touto alternativou. Jako zástupce dostupnějšího manipulátoru vhodného pro výukové účely a testování je v této práci uveden uArm Swift Pro, který je již na FEL ZČU hojně využíván v řadě projektů.

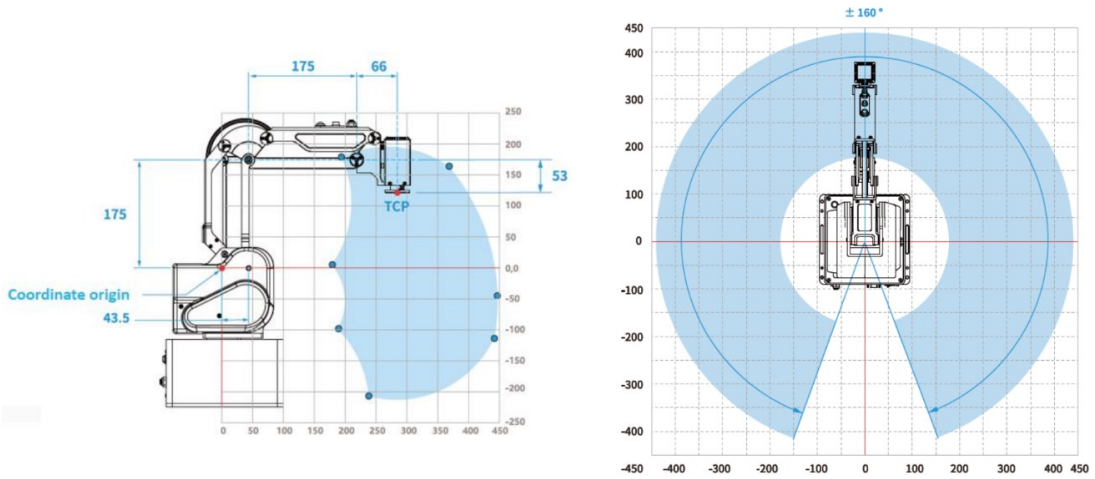
3.1 DOBOT MG400

Jedná se o stolní robotický manipulátor od firmy DOBOT a je určen převážně pro automatizaci v malovýrobě a nenáročných aplikacích. Manipulátor se skládá ze čtyř rotačních kloubů, které jsou spojeny dvěma rameny. Z toho vyplývá, že má čtyři stupně volnosti. V každém kloubu se nachází servomotor s vysoce přesným absolutním enkodérem. Na předním rameni manipulátoru je rozhraní pro připojení koncového efektoru. Nachází se zde připojení pro vakuovou pumpu, I/O port pro nástroj a tlačítko pro ruční navádění

manipulátoru, které slouží pro učení pohybů pomocí předvádění a mimo toho i pro odblokování brzd motorů po nouzovém zastavení. [7][11]

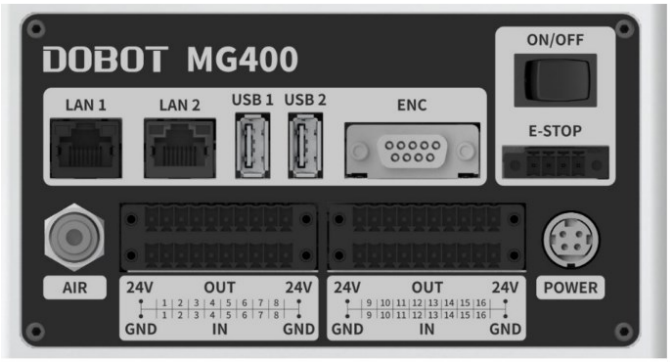
Obr. 17 DOBOT MG400 [11]

Výrobce udává opakovatelnost ±0,05 mm, maximální dosah 440 mm a nosnost 500 g, výrobce navíc udává, že maximální nosnost může být i 750 g [7]. Jako koncový efektor lze použít například vakuovou přísavku, gripper, 3D tiskovou hlavu a laser pro gravírování. Jako další příslušenství nabízí výrobce doplňkové moduly v podobě pásového dopravníku, lineárního pojezdu, robotického vidění a Arduino soupravu. Manipulátor je dále vybaven systémem pro detekci kolize, aby dokázal automaticky zastavil, pokud narazí na překážku. Tento systém slouží především pro snížení negativního dopadu na manipulátor, tak aby se předešlo poškození manipulátoru nebo jeho příslušenství. Manipulátor je připojen a komunikuje skrze LAN (Ethernet) nebo Wifi. [12]



Obr. 18 Pracovní prostor DOBOT MG400 [11]

Na zadní straně základny manipulátoru se nachází rozhraní, které obsahuje 2x Ethernet, 2x USB 2.0, port pro připojení pásového dopravníku (enkodér), rozhraní pro nouzové zastavení, port pro napájení, zdroj vzduchu pro vakuovou přísavku a I/O rozhraní pro připojení externích nástrojů v podobě 16 digitálních vstupů a 16 digitálních výstupů. [11]

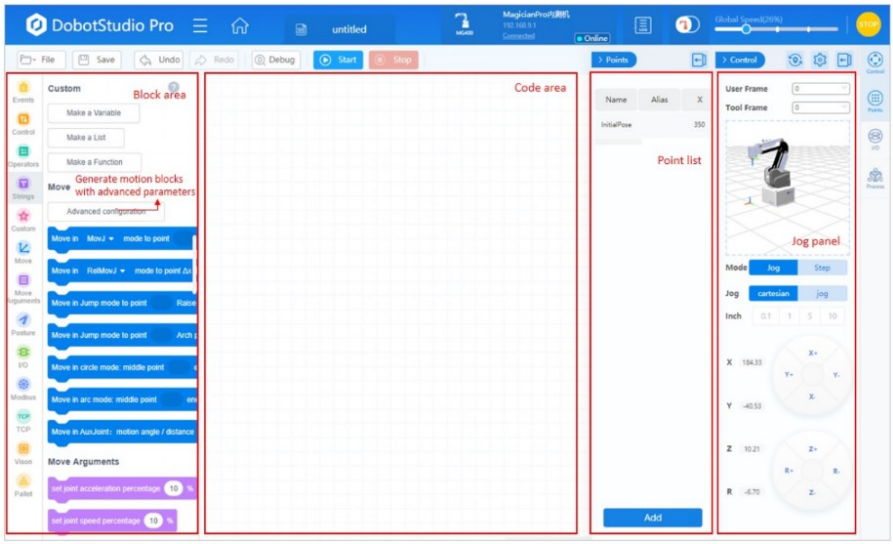


Obr. 19 Rozhraní na základně manipulátoru [12]

Mezi možné způsoby ovládání samozřejmě patří učení manipulátoru pohybům pomocí ručního navádění. Dále firma DOBOT poskytuje multifunkční řídící software DobotStudio s jednoduchým rozhraním a způsoby programování vhodných pro začátečníky. Je tedy vhodný spíše pro počáteční seznámení s manipulátorem, jeho pohyby a možnostmi. Součástí DobotStudio je vizuální programovací editor, DobotBlockly, aplikace pro 3D tisk a Script. [14]

U Blocky není potřeba znát syntaxi programovacího jazyka, ale jen základní logiku vytváření kódu. Jednotlivé příkazy jsou k sobě skládány ve formě barevných bloků. Výhodou je možnost vytvářet i komplikovanější aplikace a následně vytvořený kód vygenerovat v jazyce Python, který lze poté použít a upravovat v jiném vývojovém prostředí. [14]

V prostředí s názvem Script lze již programovat klasickým způsobem, jako je tomu u známějších alternativ v podobě VS Code a PyCharm. Výhodou tohoto prostředí oproti populárnějším možnostem je nápověda a vysvětlení jednotlivých příkazů. V levém panelu se nachází nabídka s jednotlivými kategoriemi příkazů, jejich popisem a parametry. Jedná se o různé pohybové příkazy, ovládání vstupů a výstupu, TCP/UDP příkazy atd.



Obr. 20 Programovací prostředí DobotStudio [14]

Pro vývoj složitějších aplikací budou však budou však vhodná jiná výše zmíněná programovací prostředí. Z tohoto důvodu jsou od výrobce k dispozici SDK a knihovny pro sekundární vývoj, které jsou k dispozici hned v několika různým programovacích jazycích, mezi které patří například Python, Arduino, C++, C# a Lua.

SDK neboli Software Development Kit je sada knihoven a softwarových nástrojů, které mohou vývojáři použít pro řízení pohybů manipulátoru, interagovat s jeho senzory a příslušenstvím a mít přístup k dalším funkcím hardwaru.

Tabulka 1 Parametry DOBOT MG400 [7]

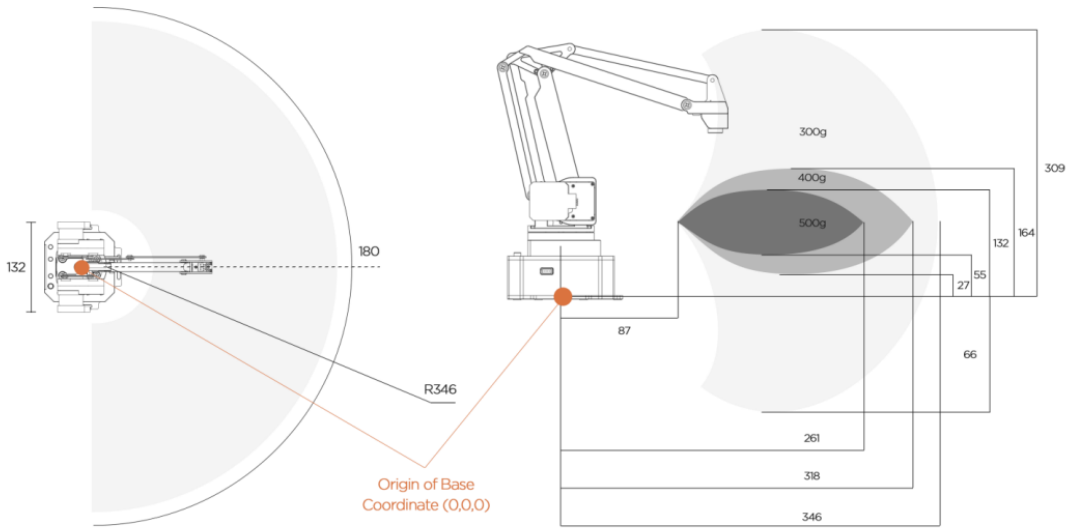
Počet stupňů volnosti		4
Nosnost		500 g (Max. 750 g)
Maximální dosah		440 mm
Opakovatelnost		±0,05 mm
Rozsah pohybů	J1	±160°
	J2	-25°~85°
	J3	-25°~105°
	J4	±180°
Maximální kloubová rychlost	J1	300°/s
	J2	300°/s
	J3	300°/s
	J4	300°/s

3.2 uArm Swift Pro

Jedná se o stolní open-source robotický manipulátor od firmy uFactory, určený převážně pro vzdělávací účely. K tomu slouží hlavně software poskytovaný výrobcem, který zahrnuje i možnost blokového programování a je tak vhodný pro úplné začátečníky a lze se na něm seznámit s principy robotiky a programování robotických manipulátorů. Manipulátor má čtyři stupně volnosti. Jedním z nich je koncový efektor, který je vždy vodorovný. To je dáno specifickou konstrukcí manipulátoru, která obsahuje i volně pohyblivé klouby a paralelogram se sekundárním přepákováním. Ramena jsou vyrobena z hliníku a celý manipulátor tak váží pouze 2,2 kg. V manipulátoru jsou použity krokové motory s kroutícím momentem 1,2 Nm s převodovkou z tvrdé oceli s nízkou vůlí. Na převodovce je 12bitový absolutní enkodér, který zajišťuje zpětnovazební určování polohy. [8]

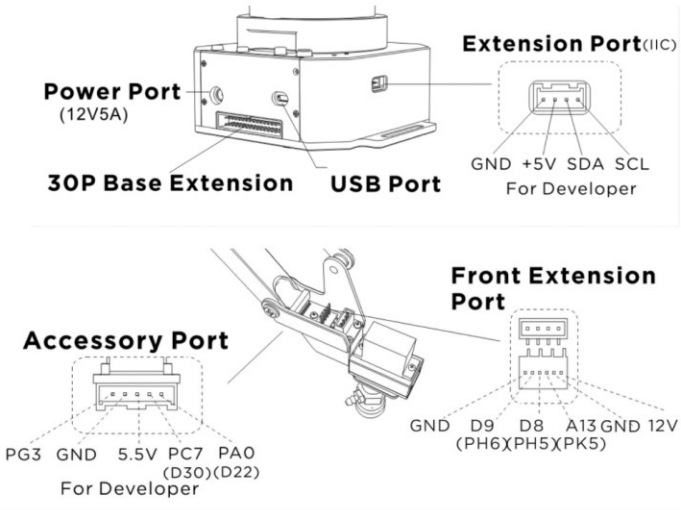
Obr. 21 uArm Swift Pro [9]

Výrobce udává opakovatelnost 0,2 mm, maximální dosah 320 mm (50~320 mm) a nosnost 500 g. Maximální zatížení je ve skutečnosti proměnné a závisí na poloze koncového efektoru v pracovním prostoru. [8]



Obr. 22 Pracovní prostor uArm Swift Pro [10]

Na základně manipulátoru se nachází micro USB port pro komunikaci s PC. Je zde i možnost bezdrátové komunikace prostřednictvím Bluetooth 4.0. Dále napájecí port, rozšiřující port pro připojení příslušenství a 30 pinový konektor pro potřeby 3D tisku. [9]

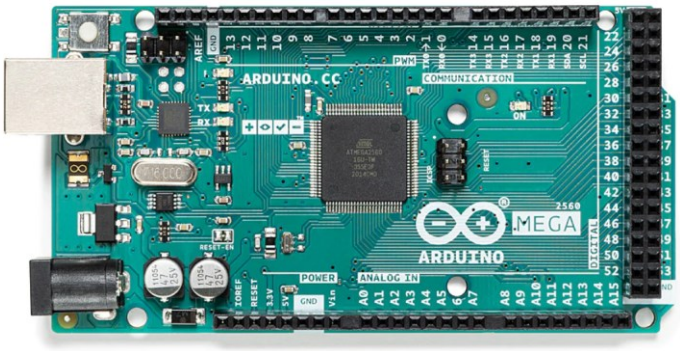


Obr. 23 Rozhraní na základně a předním rameni manipulátoru [10]

Výrobce dodává příslušenství pro přizpůsobení manipulátoru řadě aplikací. Nástroje pro koncový efektor zahrnují například vakuovou přísavku, gripper, laser a 3D tiskovou sadu. Mezi další příslušenství spadá lineární pojezd, pásový dopravník a kontrolér. [9]

Obr. 24 Nástroje pro koncový efektor [9]

Manipulátor je ovládán pomocí mikrokontroleru Arduino Mega 2560 využívající mikroprocesor ATmega2560 s 16 MHz krystalovým oscilátorem. Flash paměť pro kód činí 256 kB a dočasná RAM neboli bootloader je o velikosti 8 kB. Deska má 54 digitálních I/O pinů, kde lze 15 z nich použít jako výstupy PWM. Dále 16 analogových vstupů, připojení USB, 4 konektory UART (sériové porty), vstupní napětí je 7~12 V a deska pracuje s 5 V. Mikrokontroler je programovatelný přes vývojové prostředí Arduino Software (IDE). Právě pomocí tohoto prostředí je uskutečněná počáteční kalibrace manipulátoru a přes sériový port lze posílat příkazy v podobě G kódů. [13]



Obr. 25 Arduino Mega 2560 [13]

Pro počáteční seznámení s manipulátorem lze využít software poskytovaný výrobcem uArm Studio. Mezi možné způsoby ovládání a programování patří režim učení pomocí ručního navádění (manipulátor v učícím módu zaznamenává řadu akcí a pohybů, které poté automaticky přehrává), Blockly (ovládání manipulátoru pomocí vizuálního programovacího rozhraní, uživatel programuje manipulátor pomocí jednotlivých drag-n-drop funkčních bloků), Draw/Laser a možnost ovládat manipulátor z mobilního telefonu přes Bluetooth pomocí speciální aplikace. Pro složitější aplikace je však opět vhodné použít jiná vývojová

prostředí jako Arduino IDE, Pycharm nebo VS Code. Vzhledem k tomu, že je manipulátor koncipován jako open-source software platforma, tak výrobce poskytuje velké množství balíčků/SDK ať už v Pythonu nebo dalších programovacích jazycích jako C++, Arduino a ROS, aby byli vývojáři schopni vytvářet nové aplikace a integrovat je do svého systému. [9]

Tabulka 2 Parametry uArm Swift Pro [8]

Počet stupňů volnosti		4
Nosnost		500 g
Maximální dosah		50~320 mm
Opakovatelnost		0,2 mm
Rozsah pohybů	J1 (motor základny)	0°~200°
	J2 (levý motor)	0°~135°
	J3 (pravý motor)	0°~100°
	J4 (motor koncového efektoru)	0°~180°
Maximální kloubová rychlost	J1 (motor základny)	40°/s
	J2 (levý motor)	40°/s
	J3 (pravý motor)	40°/s
	J4 (motor koncového efektoru)	60°/s

4 Možné způsoby programování a komunikace

4.1 Komunikace na nízké úrovni (G kód)

Jedním z možných způsobů komunikace s robotickými manipulátory je pomocí zasílání G kódů. G kód je standardní jazyk, který se používá v průmyslu k řízení pohybů stroje, kde může jít například o CNC nebo právě robotický manipulátor. Jako příklad lze uvést příkaz pro jednoduchý pohyb na dané souřadnice v podobě G0 X100 Y100 Z50 F1000. Tento příkaz říká, aby manipulátor pohnul svým koncovým efektorem na dané kartézské souřadnice s určitou rychlostí a manipulátor si tento příkaz převede na pohyb jednotlivých motorů. Samozřejmě nejde jen o řízení pohybů, ale i o řízení akcí koncového efektoru a dalších funkcí. Příkladem je přepínání jednotlivých módů v závislosti na použitém koncovém efektoru, kde se liší středový bod jednotlivých nástrojů vůči počátku souřadného systému uprostřed základny. Jsou používány příkazy M2400 S0 pro použití vakuové přísavky, M2400 S3 pro univerzální držák atd. [10]

Pomocí zasílání G kódů přes sériový port je uskutečněna i počáteční kalibrace manipulátoru uArm Swift Pro. Ta slouží pro určení offsetů motorů. Nejdříve je však potřeba připojit manipulátor k počítači a v prostředí Arduino IDE zvolit správný COM port a následně nastavit přenosovou rychlost na 115200 baud. Po připojení lze začít zadávat příkazy pro kalibraci, jako M2019 pro deaktivaci motorů, a poté začít přikládat koncový efektor bez namontovaného nástroje na jednotlivé body kalibrační šablony, kde jsou zasílány příkazy v podobě M2401 A/B/C. [15]

G kód u manipulátoru uArm Swift Pro přináší další specifika, kde je na základě standardního G kód protokolu přidána hlavička pro snadnější debuggování a používání. Tento způsob je však navržen tak, aby byl stále kompatibilní se standardním G kódem. Při posílání příkazů z počítače se použije například #25 G0 X100 Y100 Z50 F1000 a odpověď manipulátoru musí být $25 ok. [10]

Další možnou variantou práce s G kódem je jeho úprava pro manipulátor uArm Swift Pro. Součástí SDK tohoto manipulátoru je modul protocol.py, kde je seznam odesílaných G kódů ve formě řetězců. Před odesláním by se tyto řetězce mohly formátovat tak, aby byly kompatibilní s jinými manipulátory. Tím by se daly řešit některé základní akce pouze přes úpravu G kódu a nebylo by nutné tuto problematiku řešit ve vysokoúrovňovém kódu.

Alternativou však může být psaní vysokoúrovňového kódu, který by abstrahoval od nízko úrovňových detailů komunikačního protokolu a příkazů v podobě G kódů. Bylo by tak možné napsat program v podobě API kompatibilního s více manipulátory.

4.2 Programování a ovládání manipulátoru pomocí ROS2

Výrobci manipulátorů nabízí i podporu ROS v podobě balíčků, které umožňují komunikaci s manipulátorem pomocí sériového portu, simulaci v RViz a plánování pohybu v MoveIt.

ROS neboli Robot Operating System slouží jako abstraktní vrstva spojující operační systém počítače s manipulátorem. Nejedná se o plnohodnotný operační systém, ale o open-source kolekci softwarových frameworků určenou pro vývoj software a správu balíčků v oblasti robotiky. ROS se dělí na jádro a robotický software. Jádro je vyvíjeno firmou Willow Garage a jednotlivé balíčky jsou vytvářeny rozsáhlou komunitou. Jádro obsahuje nástroje pro správu balíčků, jejich distribuci přes repositáře, a middleware pro propojení a kompatibilitu různého softwaru napříč platformami. [17]

ROS je založeno na technologii peer-to-peer, kde spolu komunikují jednotliví klienti. Klienti mezi sebou komunikují přes messages a services. Jednotlivé programy mohou tak běžet nezávisle na sobě na více počítačích a komunikovat přes síť. Základem struktury ROS jsou uzly master a nodes, kde nodes jsou jednoúčelové programy/jednotlivé procesy v balíčcích a speciální uzel master spravuje komunikaci mezi těmito uzly. Uzly komunikují přes tzv. topics, což je proud zpráv mezi uzly. Uzly jednotlivé topics pak buď publikují (publish), nebo odebírají (subscribe). Mezi další nástroje patří například launch, který slouží pro spouštění uzlů a nastavování parametrů. ROS obsahuje mnoho dalších komponent, kde za zmínku stojí packages, což je software ROSu organizovaný do balíčků. Každý balíček by měl obsahovat zdrojový kód, konfigurační soubory, messages a dokumentaci. Příkladem může být balíček MoveIt, což je knihovna, ve které se provádí plánování trajektorie. Součástí ROS je dále RViz, což je prostředí pro simulaci aktuálního stavu manipulátoru. Model, který obsahuje kinematický a dynamický popis, vizualizaci a kolizní model manipulátoru. Nebo Serial, což je multiplatformní knihovna pro práci se sériovými porty. [17]

Jak bylo řečeno, výrobci obou manipulátoru nabízí i podporu ROS. Nevýhodou je však, že jejich balíčky ROS podporují málo programovacích jazyků. Zatímco uArm Swift Pro podporuje C++ a Python, tak DOBOT MG400 podporuje pouze C++. To omezuje možnost vývoje a integrace s jinými nástroji a knihovnami. Navíc jsou balíčky zastaralé a nejsou dále

výrobcem aktualizovány. To znamená, že by nemusely fungovat správně s novějšími verzemi, jako ROS2 a vyžadovalo by se mnoho úprav.

4.3 Vysokoúrovňové programování v podobě API

Application Programming interface (rozhraní pro programování aplikací). Cílem API je oprostit se od detailů ovládání a programování manipulátoru na nižší úrovni a poskytnout jednodušší a uživatelsky přívětivější rozhraní pro používání funkcí softwaru nebo hardwaru a provádění běžných akcí. Zjednodušeně je to knihovna, která se skládá ze sady předdefinovaných metod a funkcí, které dovolují vývojáři ovládat manipulátor za použití vysokoúrovňových příkazů. Jde o funkce, které například obsahují příkazy pro pohyb na danou pozici, čtení dat ze senzorů, ovládání koncového efektoru a další akce. [16]

4.4 Programovací jazyk a vývojové prostředí

Pro potřeby současné testovací platformy, která je založena na Robot Framework, je nutnost implementace API v Pythonu. To je díky jeho jednoduché syntaxi velmi rozšířený vysokoúrovňový jazyk, kde je k dispozici mnoho knihoven a modulů poskytovaných výrobci pro práci s robotickými manipulátory.

Robot Framework je open-source framework pro automatizaci testování, který umožňuje vytvářet a spouštět automatizované testy pro různé typy softwarových aplikací. Je také napsaný v jazyce Python a má přehlednou a snadno čitelnou syntaxi psanou v tzv. keywords. Ty jsou definované v knihovnách, které se dají samozřejmě neustále rozšiřovat a mohou být použity pro vytváření testovacích scénářů.

Dalším krokem je volba vhodného vývojového prostředí. Jak bylo zmíněno v předešlých kapitolách u přehledu manipulátorů, software dodávaný výrobcem poskytuje jen základní funkce a omezené možnosti programování. Nutnost volby jiné alternativy pramení i z toho důvodu, že software poskytovaný výrobcem je vázán na konkrétní model manipulátoru a není tak vhodný pro vývoj aplikací, které mají pracovat s různými manipulátory nezávisle na výrobci. Díky intuitivnímu uživatelskému rozhraní bylo pro programování zvoleno prostředí Visual Studio Code, které tak umožní vývoj univerzálního API, které bude kompatibilní pro oba manipulátory.

5 API robotických manipulátorů

Při implementaci společného API je potřeba brát v potaz, že oba manipulátory mají jinou kinematiku, řídící systém, parametry a možnosti. To znamená, že jejich stávající API budou rozdílná. Proto se nabízí řešení, kde bude navrženo API pro každý manipulátor zvlášť, ovšem s podobnou strukturou a sadou běžných funkcí a metod používaných oběma manipulátory. Tyto metody by pak bylo možné jednoduše implementovat ve společném API, kde by nebylo nutné se starat o specifické detaily jednotlivých API.

Nejprve je potřeba analyzovat současné API DOBOT MG400, které již bylo upraveno a obsahuje všechny potřebné metody pro požadavky současné testovací platformy. Následně je potřeba analyzovat i API manipulátoru uArm Swift Pro a opět se zaměřit na potřebné metody. Cílem je ve finále, na základě této analýzy, sjednotit API obou manipulátorů a vytvořit nadstavbu ve formě wrapperu, který by umožnil, aby bylo jejich použití a ovládání co nejjednodušší a nejpřehlednější. API by mělo obsahovat metody, jako výběr manipulátoru, inicializace, základní pohyby, zasílání chybových hlášek, ovládání koncového efektoru atd.

5.1 Analýza API a knihoven manipulátoru DOBOT MG400

Celý program pro ovládání DOBOT MG400 (dostupný z [18]) se skládá z následujících pěti souborů:

- **main.py**: hlavní skript programu, který slouží pro připojení k manipulátoru, načtení virtuální klece a testování základních funkcí manipulátoru. Kromě toho obsahuje samotnou testovací sekvenci čtečky čárových kódů. Tento skript volá metody z **dobot_api_run.py** a **dobot_api.py**, které realizují pohyby a operace robota.
- **dobot_api_run.py**: obsahuje základní, ale i sofistikovanější metody pro ovládání manipulátoru. Obsahuje například metodu connect_robot(), která vytváří instance tříd *DobotApi, DobotApiMove a DobotApiDashboard* z **dobot_api.py** a navazuje socketovou komunikaci s manipulátorem. Dále obsahuje metody jako vir_cage_set(), která definuje meze pohybu manipulátoru podle jeho souřadnicového systému. Také obsahuje metody pro získání aktuální pozice manipulátoru get_pos(), lineární pohyb run_pointL(), kloubový pohyb run_pointJ() atd.
- **dobot_api.py**: samotné rozhraní pro komunikaci s manipulátorem. Obsahuje tři třídy: *DobotApi* – zajišťuje socketovou komunikaci s manipulátorem a posílá

a přijímá data v binárním formátu; *DobotApiDashboard* – poskytuje metody pro nastavení a kontrolu stavu manipulátoru, jako jsou EnableRobot(), SpeedFactor(), DO() atd.; *DobotApiMove* – poskytuje metody pro pohyb a operace manipulátoru, jako jsou EmergencyStop(), MovL(), MovJ() atd.

- **alarm_servo.py**: sada alarmů, které během používání manipulátoru mohou nastat. Každý alarm je definován svým id a krátkým popisem. Například alarm s id 29056 znamená "Motor is in overload status", alarm s id 34322 znamená „Position is out of range" atd.
- **alarm_controller.py**: opět sada alarmů, které během používání manipulátoru mohou nastat. Příkladem zde může být alarm s id 49672, který znamená "Input params is error", a alarm s id 49668 znamená "Connect failed" atd.

▪ **Import**

Mezi základní třídy pro ovládání manipulátoru patří *ApiRun* ze souboru **dobot_api_run.py** a *DobotApi*, *DobotApiDashboard*, *DobotApiMove* ze souboru **dobot_api.py**, které lze importovat jako:

```
import dobot_api_run
from dobot_api import DobotApiDashboard, DobotApi, DobotApiMove, MyType
```

Pro připojení manipulátoru k počítači je potřeba zavolat metodu connect_robot() z třídy *ApiRun*. Tato metoda vrací instance tříd: *DobotApiDashboard*, *DobotApiMove* a *DobotApi*.

```
dashboard, move, feed = dobot_api_run.ApiRun.connect_robot()
```

▪ **DobotApiDashboard**

▫ EnableRobot():

```
def EnableRobot(self):
```

Metoda EnableRobot() slouží k aktivaci manipulátoru a umožňuje jeho pohyb.

▫ DisableRobot():

```
def DisableRobot(self):
```

Metoda DisableRobot() slouží k deaktivaci manipulátoru a zabránění jeho pohybu.

▫ ClearError():

```
def ClearError(self):
```

Metoda ClearError() slouží k odstranění alarmů kontroléru, které by mohly ovlivnit funkci manipulátoru.

▫ Speecfactor():

```
def SpeedFactor(self, speed):
```

Metoda SpeedFactor() slouží k nastavení koeficientu rychlosti manipulátoru. Jako parametr přijímá celé číslo v rozsahu 1~100.

▫ RobotMode():

```
def RobotMode(self):
```

Metoda RobotMode() slouží ke zjištění aktuálního stavu manipulátoru. Metoda vrací celé číslo reprezentující mód manipulátoru. Může jít například o stav Enable, Running state, Alarm state atd.

▫ DO():

```
def DO(self, index, status):
```

Metoda DO() slouží k nastavení digitálního výstupu a ovládání periférií manipulátoru. Metoda přijímá dva parametry v podobě celých čísel: index a status. Index je daný digitální výstup a status je jeho stav, tzn. 0 nebo 1.

▫ CP():

```
def CP(self, ratio):
```

Metoda CP() slouží k nastavení hladkosti pohybu manipulátoru mezi body. Metoda přijímá parametr v podobě celého čísla, reprezentující daný poměr o hodnotě 1~100.

▪ **DobotApiMove**

▫ GetPos():

```
def GetPos(self):
```

Metoda GetPos() slouží k získání aktuální polohy a orientace koncového efektoru. Metoda vrací seznam hodnot, které reprezentují souřadnice v kartézském systému a úhel natočení koncového efektoru.

▫ MovL():

```
def MovL(self, x, y, z, r):
```

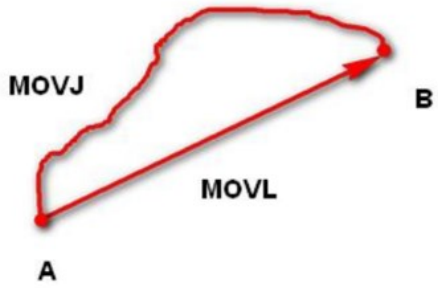
Metoda MovL() slouží k pohybu manipulátoru v lineárním režimu. Metoda přijímá čtyři parametry v podobě souřadnic v kartézském systému a úhel natočení koncového efektoru.

▫ MovJ():

```
def MovJ(self, x, y, z, r):
```

Metoda MovJ() slouží k pohybu manipulátoru v režimu point-to-point. Metoda přijímá stejné parametry, jako metoda MovL().

U point-to-point pohybů je důležité, zdali se jedná o kloubový pohyb MovJ, nebo lineární pohyb koncového efektoru MovL. V případě, že není potřeba lineární pohyb a závisí spíše na rychlosti pohybu, používá se kloubový pohyb.



Obr. 26 Pohyb MovJ a MovL [12]

▪ **DobotApi**

▫ send_data():

```
def send_data(self, string):
```

Metoda send_data() slouží k odeslání dat manipulátoru pomocí TCP/IP protokolu. Metoda přijímá parametr v podobě řetězce s příkazem pro manipulátor.

▫ wait_reply():

```
def wait_reply(self):
```

Metoda wait_reply() slouží k přijetí dat od manipulátoru pomocí TCP/IP protokolu. Metoda vrací řetězec s odpovědí manipulátoru.

▫ close():

```
def close(self):
```

Metoda close() slouží k uzavření portu pro komunikaci s manipulátorem.

5.2 Analýza API a knihoven manipulátoru uArm Swift Pro

Výrobce manipulátoru uArm Swift Pro poskytuje funkční SDK/API, které je dostupné z [19]. Zde je důležitý adresář wrapper, který obsahuje hlavní modul pro ovládání manipulátoru uArm Swift Pro pomocí Pythonu. V tomto adresáři se nachází soubor **swift_api.py**. To je modul, který definuje třídu *SwiftAPI* pro ovládání manipulátoru a pro jeho implementaci v cílové aplikaci tak stačí importovat pouze třídu *SwiftAPI*. Tato třída dědí od třídy *Swift* z adresáře swift a rozšiřuje ji o další metody a atributy. Tento modul si naimportuje všechny potřebné soubory a moduly z adresářů comm, swift, tools a utils, které obsahují různé funkce a protokoly pro komunikaci a pohyb manipulátoru.

Základem knihovny uArm-Python-SDK je tedy adresář uarm, který obsahuje hlavní soubory a moduly pro práci s manipulátorem. V tomto adresáři se nachází následující podadresáře a soubory:

- Adresář comm obsahuje modul **threaded.py** pro komunikaci s manipulátorem pomocí sériového portu, který obsahuje metody pro otevření, zavření a čtení ze sériového portu nebo pro vytvoření komunikačního vlákna a posílání a přijímání dat z manipulátoru.
- Adresář swift obsahuje moduly **gripper.py**, **grove.py**, **keys.py**, **multi.py**, **protocol.py**, **pump.py**, **teach.py** a **utils.py** pro ovládání a interakci s manipulátorem, jeho příslušenstvím a nástroji. Hlavní je zde třída *Swift*, která je následně importována v samotném **swift_api.py**. Jednotlivé moduly obsahují metody pro pohyby, výpočet polohy, ovládání nástrojů a definují konstanty a funkce pro komunikační protokol manipulátoru.
- Adresář tools obsahuje moduly **config.py**, **list_ports.py** a **threads.py**. Obsahuje nástroje pro aktualizaci firmware, výpis dostupných portů a správu vláken.
- Adresář utils obsahuje modul **log.py**, který definuje funkce pro logování zpráv do konzole nebo do souboru.
- Adresář wrapper obsahuje modul **swift_api.py**, který slouží pro zjednodušení použití SDK pomocí vyšší úrovně abstrakce. Definuje třídu *SwiftAPI*, která umožňuje ovládání manipulátoru pomocí jednoduchých příkazů.
- Soubor **version.py** definuje verzi knihovny uArm-Python-SDK.

Stejně jako u DOBOT MG400, i zde obsahuje API manipulátoru mnoho metod a akcí, které může manipulátor provádět. Opět zde budou uvedeny jen ty metody, které jsou relevantní pro ovládání manipulátoru při testování čtečky pomocí přikládání karet s čárovými kódy. Jednotlivé metody lze rozdělit do následujících kategorií:

▪ **Import**

Základní třídou pro ovládání manipulátoru je *SwiftAPI*, kterou lze importovat jako:

```
from uarm.wrapper import SwiftAPI
```

Pro vytvoření instance třídy *SwiftAPI*, která umožňuje ovládat manipulátor pomocí API, je potřeba zavolat konstruktor:

```
swift = SwiftAPI(filters={'hwid': 'USB VID:PID=2341:0042')
```

Konstruktor přijímá několik volitelných parametrů, které nastavují vlastnosti komunikace a pohybu manipulátoru. Parametry jsou port, baudrate, timeout a filters. Parametr port je název sériového portu, na kterém je manipulátor připojen. Pokud není zadán, konstruktor se pokusí najít první dostupný port. Parametr baudrate udává rychlost přenosu dat v bit/s. Pokud není zadán, použije se výchozí hodnota 115200 bit/s. Parametr timeout udává maximální dobu čekání na odpověď od manipulátoru v sekundách. Pokud není zadán, použije se výchozí hodnota 2 s. Parametr filters je seznam filtrů, které se aplikují na data získaná z manipulátoru.

▪ **Connect a disconnect**

▫ connect():

```
def connect(self, port=None, baudrate=None, timeout=None):
```

Metoda connect() slouží k navázání komunikace mezi manipulátorem a počítačem pomocí sériového portu. Metoda přijímá tři volitelné parametry: port, baudrate a timeout. Parametr port určuje název sériového portu, na kterém je manipulátor připojen. Parametr baudrate určuje rychlost přenosu dat v bit/s. Parametr timeout určuje maximální dobu čekání na odpověď od manipulátoru v sekundách. Pokud není jeden z těchto parametrů zadán, použije se hodnota zadaná při inicializaci. Metoda vrací True, pokud se podařilo úspěšně připojit k manipulátoru, nebo False, pokud došlo k chybě.

▫ disconnect():

```
def disconnect(self, is_clean=True):
```

Metoda disconnect() slouží k ukončení komunikace mezi manipulátorem a počítačem. Metoda přijímá jeden volitelný parametr: is_clean. Ten určuje, zda se má vyčistit thread

pool, ve kterém jsou vlákna používána pro komunikaci s manipulátorem. Pokud je True, thread pool se vyčistí, a naopak při False. Metoda vrací True, pokud se podařilo úspěšně odpojit od manipulátoru, nebo False, pokud došlo k chybě.

- **Get**
  - get_position():

```
def get_position(self, wait=True, timeout=None, callback=None):
```

Metoda get_position() slouží k získání aktuální polohy manipulátoru v kartézských souřadnicích. Metoda přijímá tři volitelné parametry: wait, timeout a callback). Parametr wait určuje, zda se má čekat na odpověď od manipulátoru. Pokud je True, metoda čeká na odpověď. Pokud je False, metoda okamžitě vrátí None. Parametr timeout opět určuje maximální dobu čekání na odpověď manipulátoru. Parametr callback určuje funkci, která se zavolá po obdržení odpovědi od manipulátoru. Pokud není zadán, nepoužije se žádná funkce. Metoda vrací seznam [x, y, z], což jsou souřadnice polohy manipulátoru v milimetrech, pokud se podařilo úspěšně získat polohu. Pokud dojde k překročení timeoutu, metoda vrací řetězec ‘TIMEOUT‘.

  - get_polar():

```
def get_polar(self, wait=True, timeout=None, callback=None):
```

Metoda get_polar() slouží k získání aktuální polohy manipulátoru v polárních souřadnicích. Metoda přijímá tři volitelné parametry: wait, timeout a callback. Oproti metodě get_position() je rozdíl pouze v návratové hodnotě, kde tato metoda vrací seznam [stretch, rotation, height], kde stretch je vzdálenost koncového efektoru od středu základny v milimetrech, rotation je úhel otočení koncového efektoru kolem osy z a height je výška koncového efektoru nad základnou.

  - get_servo_angle():

```
def get_servo_angle(self, servo_id=None, wait=True, timeout=None, callback=None):
```

Metoda get_servo_angle() slouží k získání úhlu natočení servomotorů manipulátoru. Metoda přijímá čtyři volitelné parametry: servo_id, wait, timeout a callback. Parametr servo_id určuje identifikátor servomotoru, jehož úhel chceme získat. Pokud není zadán, metoda vrátí seznam úhlů natočení všech servomotorů. Identifikátory servomotorů jsou: 0 pro motor základny, 1 pro levý motor a 2 pro pravý motor. Ostatní parametry jsou totožné s předešlými metodami.

- get_mode():

```
def get_mode(self, wait=True, timeout=None, callback=None):
```

Metoda get_mode() slouží k získání aktuálního režimu manipulátoru. Metoda přijímá tři volitelné parametry: wait, timeout a callback. Metoda vrací režim manipulátoru jako celé číslo. Režimy jsou: 0 pro normální režim, 1 pro laserový režim, 2 pro 3D tiskový režim a 3 pro univerzální držák.

- get_pump_status():

```
def get_pump_status(self, wait=True, timeout=None, callback=None):
```

Metoda get_pump_status() slouží k získání stavu čerpadla vzduchu pro vakuovou přísavku. Metoda přijímá tři volitelné parametry: wait, timeout a callback. Metoda vrací stav čerpadla jako celé číslo. Stavy jsou: 0 pro zastavené čerpadlo, 1 pro pracující čerpadlo a 2 v případě, že je nasán objekt.

▪ Set

- set_pump():

```
def set_pump(self, on=False, timeout=None, wait=True, check=False, callback=None):
```

Metoda set_pump() slouží k ovládání čerpadla vzduchu pro vakuovou přísavku. Metoda přijímá pět volitelných parametrů: on, wait, check, timeout a callback. Parametr on určuje, zda se má zapnout nebo vypnout čerpadlo. Pokud je True, čerpadlo se zapne. Výchozí hodnota je False. Parametr check určuje, zda se má zkontrolovat stav čerpadla po potvrzení od manipulátoru. Pokud je True, metoda zkontroluje stav čerpadla a vrátí ‘OK‘, pokud se shoduje s požadovaným stavem, nebo ‘TIMEOUT‘, pokud se neshoduje. Pokud je False, metoda nekontroluje stav čerpadla a vrátí ‘OK‘, pokud obdrží potvrzení od manipulátoru, nebo ‘TIMEOUT‘, pokud nepřijde potvrzení v daném čase. Výchozí hodnota je False.

- set_mode():

```
def set_mode(self, mode=0, wait=True, timeout=None, callback=None):
```

Metoda set_mode() slouží k nastavení režimu manipulátoru. Metoda má čtyři volitelné parametry: mode, wait, timeout a callback. Parametry a možnosti režimů jsou stejné jako u metody get_mode().

▫ set_acceleration():

```
def set_acceleration(self, acc=None, wait=True, timeout=None, callback=None):
```

Metoda set_acceleration() slouží k nastavení zrychlení manipulátoru. Metoda přijímá čtyři volitelné parametry: acc, wait, timeout a callback. Parametr acc určuje hodnotu zrychlení.

▫ set_speed_factor():

```
def set_speed_factor(self, factor=1):
```

Metoda set_speed_factor() slouží k nastavení koeficientu rychlosti manipulátoru. Rychlost pohybu manipulátoru se vynásobí tímto koeficientem. Metoda přijímá jeden volitelný parametr: factor. Tento parametr určuje právě hodnotu koeficientu rychlosti. Výchozí hodnota je 1. Metoda nevrací žádnou hodnotu.

▪ **Event register/release**

▫ register_power_callback():

```
def register_power_callback(self, callback=None):
```

Metoda register_power_callback() slouží k nastavení funkce, která se zavolá při změně stavu napájení manipulátoru. Metoda přijímá jeden volitelný parametr: callback. Ten určuje funkci, která se volá při změně stavu napájení.

▫ release_power_callback():

```
def release_power_callback(self, callback=None):
```

Metoda release_power_callback() slouží k uvolnění funkce, která se zavolá při změně stavu napájení manipulátoru. Metoda přijímá jeden volitelný parametr: callback. Ten určuje funkci, která se má uvolnit. Pokud není zadán, uvolní se všechny funkce zaregistrované pomocí metody register_power_callback().

▫ register_limit_switch_callback():

```
def register_limit_switch_callback(self, callback=None):
```

Metoda register_limit_switch_callback() slouží k nastavení funkce, která se zavolá při změně stavu koncových spínačů manipulátoru. Metoda přijímá jediný volitelný parametr: callback. Parametr callback určuje funkci, která se zavolá při změně stavu.

▫ release_limit_switch_callback():

```
def release_limit_switch_callback(self, callback=None):
```

Metoda release_limit_switch_callback() slouží k uvolnění funkce, která se zavolá při změně stavu koncových spínačů manipulátoru. Metoda přijímá jeden volitelný parametr: callback. Ten určuje funkci, která se má uvolnit. Pokud není zadán, uvolní se všechny funkce zaregistrované pomocí metody register_limit_switch_callback().

▪ **Wait**

▫ waiting_ready():

```
def waiting_ready(self, timeout=5, **kwargs):
```

Metoda waiting_ready() slouží k čekaní na to, až bude manipulátor připraven k provádění příkazů. Metoda přijímá dva parametry: timeout a **kwargs. Výchozí hodnota parametru timeout, který určuje maximální dobu čekání, je 5 s. Parametr **kwargs umožňuje předat další argumenty do metody. Metoda vrací True, pokud je dokončen pohyb a manipulátor je opět připraven.

▫ flush_cmd():

```
def flush_cmd(self, timeout=None, wait_stop=False):
```

Metoda flush_cmd() slouží k čekání na dokončení všech asynchronních příkazů, které byly odeslány manipulátoru. Metoda přijímá dva parametry: timeout a wait_stop. Parametr timeout určuje dobu čekání na všechny asynchronní příkazy. Parametr wait_stop určuje, zda se má čekat na to, až se manipulátor přestane pohybovat, či nikoli. Výchozí hodnota je False (nečeká se na pohyb). Metoda vrací ‘OK‘, pokud se všechny asynchronní příkazy dokončily, nebo ‘TIMEOUT‘, pokud došlo k vypršení daného časového limitu.

▪ **Move**

▫ set_position():

```
def set_position(self, x=None, y=None, z=None, speed=None, relative=False, wait=False, timeout=10, callback=None, cmd='G0'):
```

Metoda set_position() slouží k nastavení pozice manipulátoru v kartézském souřadnicovém systému. Metoda přijímá devět parametrů: x, y, z, speed, relative, wait, timeout, callback a cmd. Parametry x, y, z určují souřadnice cílové pozice v milimetrech, výchozí hodnoty jsou poslední použité souřadnice nebo 150, 0, 150. Parametr speed určuje rychlost pohybu v milimetrech za minutu. Výchozí hodnota

je opět poslední použitá rychlost nebo 1000. Parametr relative určuje, zda se má použít relativní nebo absolutní pozicování. Výchozí hodnota je False (absolutní). Parametr wait určuje, zda se má čekat na dokončení pohybu. Výchozí hodnota je False (nečeká se). Parametr callback určuje funkci, která se má zavolat po dokončení pohybu. Parametr cmd určuje typ pohybu: G0 pro rychlý pohyb nebo G1 pro lineární interpolaci. Výchozí hodnota je G0.

- set_servo_angle():

```
def set_servo_angle(self, servo_id=0, angle=90, wait=False, timeout=10, speed=None, callback=None):
```

Metoda set_servo_angle() slouží k nastavení úhlu natočení jednoho ze čtyř motorů manipulátoru. Metoda přijímá šest parametrů: servo_id, angle, wait, timeout, speed a callback. Parametr servo_id určuje identifikátor motoru, který se má ovládat. Výchozí hodnota je 0 (motor základny manipulátoru). Parametr angle určuje cílový úhel natočení. Výchozí hodnota je 90 °.

- reset():

```
def reset(self, speed=None, wait=True, timeout=None, x=200, y=0, z=150):
```

Metoda reset() slouží k vrácení manipulátoru do výchozí pozice. Metoda přijímá šest parametrů: speed, wait, timeout, x, y, z. Parametry x, y, z určují souřadnice výchozí pozice, kde jsou výchozí hodnoty 200, 0, 150.

- check_pos_is_limit():

```
def check_pos_is_limit(self, pos=None, is_polar=False, wait=True, timeout=None, callback=None):
```

Metoda check_pos_is_limit() slouží k ověření, zda je daná pozice v dosahu manipulátoru. Metoda přijímá pět parametrů: pos, is_polar, wait, timeout a callback. Parametr pos určuje souřadnice pozice. Pokud je parametr is_polar False (výchozí hodnota), pak jsou použity kartézské souřadnice [x, y, z]. Pokud je parametr True, použijí se polární souřadnice [stretch, rotation, height]. Metoda vrací True, pokud je pozice v dosahu.

5.3 Implementace API manipulátorů DOBOT MG400 a uArm Swift Pro

Při implementaci společného API (dostupné z [20]) jsem se zaměřil jen na základní metody důležité pro cílovou aplikaci, ty jsou zahrnuty v modulu **RobotAPI.py**. Tento modul definuje třídu *RobotAPI* a obsahuje všechny metody potřebné pro ovládání manipulátoru. Pro jeho implementaci v cílové aplikaci stačí importovat třídu *RobotAPI*. Dále je potřeba importovat *ManipulatorType*, což je třída pro výčet typů manipulátorů, které jsou podporovány v aplikaci.

V následujících odstavcích bude uveden výčet metod zahrnutých ve společném API, které jsou relevantní pro ovládání manipulátoru při testování čtečky pomocí přikládání karet s čárovými kódy. U jednotlivých metod je opět uveden jejich krátký popis a metody, které z daných API volají. Vstupní parametry metod a hodnoty, které jednotlivé metody vrací jsou popsány v kapitolách s analýzou API manipulátorů DOBOT MG400 a uArm Swift Pro. U vstupních parametrů byla snaha co nejvíce parametrů sjednotit, aby byly při volání jednotlivých metod univerzální pro oba manipulátory. Parametry, které jsou specifické přímo pro daný manipulátor jsou označeny přeponou se jménem daného manipulátoru.

- select_manipulator():

```
def select_manipulator(self, manipulator: ManipulatorType):
```

Metoda select_manipulator() slouží k výběru konkrétního manipulátoru a jako vstupní parametr přijímá typ manipulátoru v podobě ManipulatorType.uArmSwiftPro, nebo ManipulatorType.dobotMG400. Metoda na základě výběru vytváří instanci třídy daného manipulátoru, která ho umožňuje ovládat pomocí API.

- connect():

```
def connect(self, uArmSwiftPro_port=None, uArmSwiftPro_baudrate=None, uArmSwiftPro_timeout=None):
```

Metoda connect() slouží k navázání komunikace mezi manipulátorem a počítačem. Na základě výběru manipulátoru volá metodu connect() pro uArm a metodu connect_robot() pro DOBOT.

- disconnect():

```
def disconnect(self, uArmSwiftPro_is_clean=True):
```

Metoda disconnect() slouží k ukončení komunikace mezi manipulátorem a počítačem. Pro manipulátor uArm je volána metoda disconnect() a pro DOBOT metoda myDisableRobot().

- set_speed_factor():

```
def set_speed_factor(self, speed=None):
```

Metoda set_speed_factor() slouží pro nastavení koeficientu, kterým se násobí rychlost manipulátoru. Pro uArm je volána metoda set_speed_factor() a pro DOBOT je volána metoda SpeedFactor().

- reset():

```
def reset(self,  x=200, y=0, z=150, uArmSwiftPro_speed=None, uArmSwiftPro_wait=True, uArmSwiftPro_timeout=None,
          dobotMG400_r=None, dobotMG400_SpF=None, dobotMG400_ConP=None):
```

Metoda reset() slouží k vrácení manipulátoru do jeho počáteční polohy. Pro uArm je volána metoda reset() a pro DOBOT je toto realizováno pomocí metody run_pointJ().

- get_mode():

```
def get_mode(self, uArmSwiftPro_wait=True, uArmSwiftPro_timeout=None, uArmSwiftPro_callback=None):
```

Metoda get_mode() slouží k získání módu manipulátoru. Pro uArm je volána metoda get_mode() a pro DOBOT metoda robot_mode_print().

- get_position():

```
def get_position(self, uArmSwiftPro_wait=True, uArmSwiftPro_timeout=None, uArmSwiftPro_callback=None):
```

Metoda get_position() slouží k získání aktuální pozice koncového efektoru. Pro uArm je volána metoda get_position() a pro DOBOT metoda get_pos().

- set_position():

```
def set_position(self, x=None, y=None, z=None, uArmSwiftPro_speed=None, uArmSwiftPro_relative=False, uArmSwiftPro_wait=False,
                 uArmSwiftPro_timeout=10, uArmSwiftPro_callback=None, uArmSwiftPro_cmd='G0',
                 dobotMG400_r=None, dobotMG400_h1=None, dobotMG400_h2=None, dobotMG400_SpF=None, dobotMG400_ConP=None):
```

Metoda set_position() slouží k nastavení pozice koncového efektoru. Pro uArm je volána metoda set_position() a pro DOBOT metoda run_point_jumpSharpL().

- set_pump():

```
def set_pump(self, on=False, uArmSwiftPro_timeout=None, uArmSwiftPro_wait=True, uArmSwiftPro_check=False,
             uArmSwiftPro_callback=None, dobotMG400_index=None):
```

Metoda set_pump() slouží k ovládání koncového efektoru v podobě vakuové přísavky. Pro uArm je volána metoda set_pump() a pro DOBOT metoda DO().

- close():

```
def close(self, uArmSwiftPro_timeout=None, uArmSwiftPro_wait_stop=False):
```

Metoda close() slouží u manipulátoru uArm k čekání na dokončení všech asynchronních příkazů, které byly manipulátoru odeslány, a volá metodu flush_cmd(). U manipulátoru DOBOT je volána metoda close(), která slouží k uzavření portu pro komunikaci s manipulátorem.

- check_pos_is_limit():

```
def check_pos_is_limit(self, uArmSwiftPro_pos=None, uArmSwiftPro_is_polar=False, uArmSwiftPro_wait=True,
                       uArmSwiftPro_timeout=None, uArmSwiftPro_callback=None):
```

Metoda check_pos_is_limit() slouží ke kontrole, zda je požadovaná pozice koncového efektoru v povolených mezích. Pro manipulátor uArm je volána metoda check_pos_is_limit() a pro DOBOT metoda vir_cage_check().

▫ enable_robot():

```
def enable_robot(self):
```

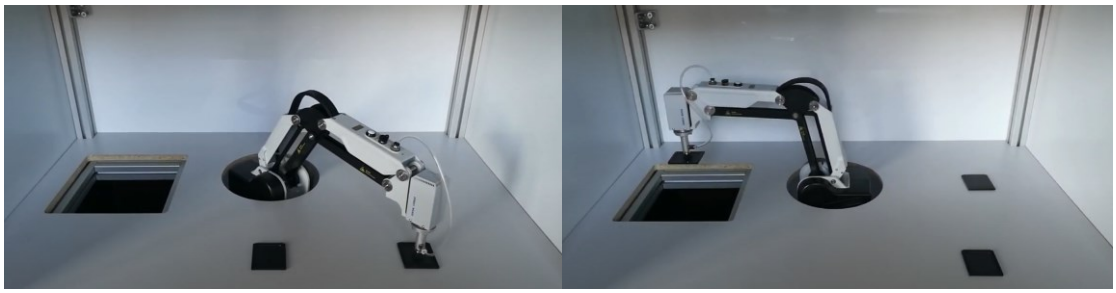
Metodu enable_robot() je potřeba volat jen při práci s manipulátorem DOBOT a slouží k aktivaci manipulátoru a umožnit tak jeho pohyb.

6 Testování navrženého API

Pro testování navrženého API je na jeho základě vytvořen jednoduchý testovací skript, který slouží pro odzkoušení a otestování základních pohybů, chybových hlášek/stavů, ovládání koncového efektoru atd. Skript obsahuje celou testovací sekvenci pro přikládání karet s čárovými kódy ke čtečce čárových kódů.

Jednotlivé karty jsou vždy rozmístěné na pevně daných pozicích, ze kterých je manipulátor postupně vyzvedává a přikládá ke čtečce. Jednotlivé pozice pak stačí nastavit jako proměnné, které obsahují souřadnice jednotlivých karet. Dále je potřeba pevně nastavit pozici čtečky a počáteční polohu manipulátoru. K rozběhnutí testu je potřeba nastavit i parametry, jako rychlost manipulátoru a relativní výška, ve které se má manipulátor pohybovat při manipulaci s objekty.

Pomocí metody select_manipulator() je vybrán požadovaný manipulátor a následně lze používat metody, jako connect(), disconnect() a close() pro navázání a ukončení komunikace. Pro ovládání koncového efektoru v podobě vakuové přísavky a jeho pozice slouží metody reset(), set_position() a set_pump(). V případě potřeby jsou k dispozici i metody set_speed_factor(), get_mode(), get_position() a check_pos_is_limit().



Obr. 27 Ukázka finální aplikace s manipulátorem DOBOT MG400

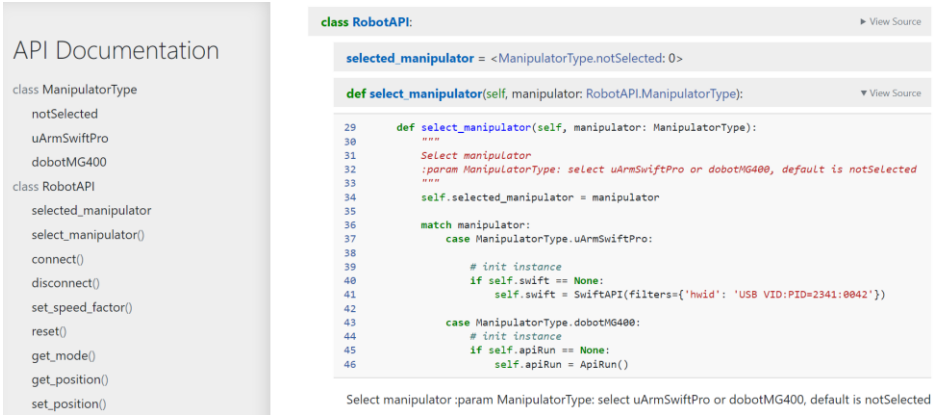
7 Dokumentace

Dokumentace slouží pro popis jednotlivých metod, kde jsou uvedeny detaily použití jednotlivých funkcí, jaké parametry přijímají, a v jakém formátu nebo jaká data API vrací. Cílem je, aby se byl vývojář nebo tester schopen v dokumentaci snadno orientovat a bylo pochopitelné, jak za pomocí API interagovat s manipulátorem, a tak ho snadno použít a integrovat do své aplikace.

Dokumentace je navržena tak, aby byla přehledná, srozumitelná a obsahovala všechny potřebné informace pro cílového uživatele. V dokumentaci jsou uvedené základní metody společné pro oba manipulátory. V případě potřeby lze rozšířit dokumentaci i o metody specifické jen pro daný manipulátor. Vzhledem k tomu, že jsou manipulátory odlišné a zadávané parametry nemusí být jednotné, jsou u společných metod uvedeny omezení a zvláštnosti pro každý manipulátor.

Dokumentace je automaticky vygenerována z hotového kódu ve formátu .html, tak aby mohla být při změně kódu, nebo rozšiřování API o další metody jednoduše aktualizována. Pro automatické vygenerování dokumentace byl použit nástroj pdoc. Ten je určen primárně pro generování API dokumentace pro projekty v jazyce Python. Pdoc vytváří HTML soubory, které obsahují popis funkcí, tříd, proměnných a dalších prvků kódu. Pdoc podporuje i Markdown formátování, typové anotace a další funkce Pythonu. Pdoc byl zvolen, protože je jednoduchý na použití, nevyžaduje žádnou konfiguraci a lze ho spustit z příkazové řádky.

Dokumentace je v podobě metod, které lze použít pro ovládání a řízení manipulátorů. Po rozkliknutí jednotlivých metod se zobrazí jejich krátký popis, a jaké parametry pro daný manipulátor přijímají. Dokumentace je dostupná z [20].



Obr. 28 Dokumentace vygenerovaná ve formátu .html

Zhodnocení a závěr

První část práce je věnována teorii ohledně robotických manipulátorů. Zde jsem se zabýval jejich konstrukcí, komponenty, řízením a plánováním pohybu. Tento postup jsem zvolil, abych byl při implementaci univerzálního API, použitelného více typy manipulátorů, schopen se orientovat ve specifických požadavcích různých typů manipulátorů a problematice jejich řízení.

Původně se měla práce zabývat návrhem a implementací API manipulátoru DOBOT MG400. Vzhledem k jeho špatné dostupnosti se však hledala vhodnější alternativa. Tou měl být zpočátku DOBOT Magician, který měl zároveň sloužit pro případné reverzní inženýrství. Nicméně se muselo kvůli problémům s dodavatelem, nevhodnému API a celkové orientaci na výuku pro ZŠ a SŠ přistoupit k další variantě v podobě manipulátoru uArm Swift Pro. Tento manipulátor byl zvolen právě z toho důvodu, že je parametry podobný verzi Magician a výrobce poskytuje dobře napsané API v jazyce Python.

V práci jsem se dále zabýval parametry, způsoby řízení a komunikace obou manipulátorů, abych byl schopen vybrat vhodný způsob finální implementace společného řízení. Zde jsem z možných způsobů komunikace a programování přistoupil k variantě návrhu společného API kompatibilního s více manipulátory. To mi dovolilo abstrahovat od nízko úrovňových detailů komunikačního protokolu v podobě G kódů.

V praktické části práce se zabývám analýzou současných API obou manipulátorů. Cílem zde bylo pochopit, jakým způsobem jsou oba manipulátory řízeny a zaměřit se na klíčové prvky a všechny potřebné metody, abych mohl tyto informace zohlednit při implementaci společného API.

Při návrhu bylo nutné brát v potaz rozdíly obou manipulátorů, ať už se jedná o rozdílnou kinematiku, řídící systém, parametry nebo jejich možnosti. Při implementaci společného API bylo využito hlavně dobře napsané API manipulátoru uArm a na jeho základě byl navržen koncept nového API, jakožto nadstavby v podobě wrapperu. Vytvořené API je ve formě základních metod pro ovládání obou manipulátorů, kde jsem se soustředil na sjednocení co nejvíce parametrů, aby byly metody univerzální pro více typů manipulátorů. Díky tomu, že byl při vytváření společného API kladen důraz především na správný koncept, je toto API jednoduše rozšiřitelné o další metody a možnost přidání dalších typů manipulátorů. Navržené API obsahuje metody pro výběr manipulátoru, jeho inicializaci, základní pohyby a ovládání koncového efektoru, tak aby na jeho základě mohl být vytvořen testovací skript.

Otestování navrženého API proběhlo vzhledem k časovým možnostem na manipulátoru uArm Swift Pro. Zde byl otestován základ API pomocí jednoduchého skriptu obsahujícího testovací sekvenci čtečky čárových kódů.

Dokumentace API byla automaticky vygenerována pomocí nástroje pdoc, tak aby mohla být v případě potřeby jednoduše aktualizována nebo rozšířena o nové metody. Vytvořená dokumentace obsahuje základní popis jednotlivých metod a jejich vstupních parametrů.

V budoucnu je samozřejmě možnost se zaměřit na rozšíření API o další metody a podporu více typů manipulátorů. Zde je potřeba vyřešit, jakým způsobem by bylo vhodné implementovat metody specifické jen pro dané manipulátory. Dále by bylo možné zaměřit se na rozšíření způsobu komunikace pomocí G kódů, pomocí kterých manipulátor komunikuje s počítačem. Zde se nabízí možnost formátování řetězců tak, aby byly kompatibilní i s jinými manipulátory. Tím by se daly řešit některé základní akce pouze přes úpravu G kódů. Další variantou je možnost implementace ROS2, kde by se jednalo o napsání nových modulů pro příslušné manipulátory.

Literatura

[1] Robot arm: Definition, components, types and the way to choose a right robot. *UFACTORY Official Website* [online]. [cit. 2023-05-13]. Dostupné z: https://www.ufactory.cc/post/robot-arm-definition-components-types-and-ways-to-choose-a-right-robot

[2] Roboti: Jak Funguje? *KUKA AG* [online]. [cit. 2023-05-13]. Dostupné z: https://www.kuka.com/cs-cz/produkty,-služby/robotické-systémy/robot-clanek#:~:text=Jak%20takový%20robot%20funguje%3F,jeho%20sensorů%20a%20měniče%20rychlosti

[3] SMUTNÝ, Vladimír. Robotika. *Úvod do kinematiky* [online]. [cit. 2023-05-13]. Dostupné z: http://cmp.felk.cvut.cz/cmp/courses/ROB/roblec/kinematikacz.pdf

[4] LYNCH, Kevin M. a PARK, Frank C. *Modern Robotics: Mechanics, planning, and Control*. Cambridge, UK: Cambridge University Press, 2017. ISBN 978-1107156302.

[5] SICILIANO, Bruno, Lorenzo SCIAVICCO, Luigi VILLANI a Giuseppe ORIOLO. *Robotics* [online]. London: Springer London, 2009 [cit. 2023-05-13]. Advanced Textbooks in Control and Signal Processing. ISBN 978-1-84628-641-4. Dostupné z: doi:10.1007/978-1-84628-642-1

[6] CRAIG, John J. *Introduction to robotics: Mechanics and control*. Upper Saddle River, NJ: Pearson Prentice Hall, 2005. ISBN 0-13-123629-6.

[7] *DOBOT MG400 Robotic Arm | Adaptable for Small Batch Production* [online]. [cit. 2023-05-13]. Dostupné z: https://www.dobot-robots.com/products/desktop-four-axis/mg400.html

[8] *uArm Swift & uArm Swift Pro Specifications* [online]. [cit. 2023-05-13]. Dostupné z: http://download.ufactory.cc/docs/en/uArm-Swift-Specifications-en.pdf

[9] *uArm Swift Pro User Manual-V4.10.2* [online]. [cit. 2023-05-13]. Dostupné z: https://www.ufactory.cc/wp-content/uploads/2023/04/uArm-Swift-Pro-User-Manual-V4.10.2.pdf

[10] *uArm Swift Pro Developer Manual-V1.0.6* [online]. [cit. 2023-05-13]. Dostupné z: https://www.ufactory.cc/wp-content/uploads/2023/04/uArm-Swift-Pro-Developer-Manual-V1.0.6.pdf

[11] *Dobot MG400 Hardware User Guide* [online]. [cit. 2023-05-13]. Dostupné z: https://www.dobot-robots.com/service/download-center?keyword=&products%5B%5D=51

[12] *Dobot MG400 Hardware User Guide V1.1* [online]. [cit. 2023-05-13]. Dostupné z: https://www.dobot-robots.com/service/download-center?keyword=&products%5B%5D=51

[13] Arduino Mega 2560 REV3. *Arduino Official Store* [online]. [cit. 2023-05-13]. Dostupné z: https://store.arduino.cc/products/arduino-mega-2560-rev3

[14] *DobotStudio Pro User Guide (MG400&M1 Pro)v2.6* [online]. [cit. 2023-05-13]. Dostupné z: https://www.dobot-robots.com/service/download-center?keyword=&products%5B%5D=51

[15] *uArm Swift Pro Calibration V1.0.2* [online]. [cit. 2023-05-13]. Dostupné z: https://www.ufactory.cc/wp-content/uploads/2023/04/uArm_Swift_Pro_Calibration_V1.0.2_20181212.pdf

[16] Lesson: What is an API? *STEMRobotics* [online]. [cit. 2023-05-13]. Dostupné z: https://stemrobotics.cs.pdx.edu/node/4221%3Froot=4196.html

[17] Robot Operating System Wiki. *ros.org* [online]. [cit. 2023-05-13]. Dostupné z: http://wiki.ros.org/Documentation
[18] Dobot_MG400. *GitLab* [online]. [cit. 2023-05-13]. Dostupné z: https://gitlab.fel.zcu.cz/projekty/rur/dobot_mg400
[19] UARM-Python-SDK. *GitHub* [online]. [cit. 2023-05-13]. Dostupné z: https://github.com/uArm-Developer/uArm-Python-SDK
[20] DP_Hradek_Filip_Robot_API. *GitLab* [online]. [cit. 2023-05-13]. Dostupné z: https://gitlab.fel.zcu.cz/zaverecne_prace/dp_hradek_filip

Seznam obrázků