TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Ovládání Universal Robotu z nadřízeného programu

Bakalářská práce

Studijní program: B2612 Elektrotechnika a informatika
Studijní obor: Elektronické informační a řídicí systémy

Autor práce: **Pavel Müller**
Vedoucí práce: doc. Ing. Josef Černohorský, Ph.D.
Ústav mechatroniky a technické informatiky

Liberec 2022

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Zadání bakalářské práce

Ovládání Universal Robotu z nadřízeného programu

Jméno a příjmení: **Pavel Müller**
Osobní číslo: M18000016
Studijní program: B2612 Elektrotechnika a informatika
Studijní obor: Elektronické informační a řídicí systémy
Zadávající katedra: Ústav mechatroniky a technické informatiky
Akademický rok: **2020/2021**

Zásady pro vypracování:

1. Seznamte se s možnostmi programováni robotů Universal Robot, včetně možností skriptovacího jazyka.
2. Vyberte vhodné rozhraní a programovací nástroje.
3. Realizujte základní ovládání robota pomocí externího software.
4. Ověřte na demonstrační úloze.

Rozsah grafických prací:	dle potřeby dokumentace
Rozsah pracovní zprávy:	30–40 stran
Forma zpracování práce:	tištěná/elektronická
Jazyk práce:	Čeština

Seznam odborné literatury:

[1] Schilling R. J. *Fundamentals of Robotics. Analysis and Control. Prentice Hall, Englewood Cliffts, New Jersey,1990*.

[2] Angeles J. *Fundamentals of Robotics Mechanical Systems. Springer-Verlag, New York, 2003,second edition*.

Vedoucí práce:	doc. Ing. Josef Černohorský, Ph.D. Ústav mechatroniky a technické informatiky
Datum zadání práce:	9. října 2020
Předpokládaný termín odevzdání:	16. května 2022

L.S.

prof. Ing. Zdeněk Plíva, Ph.D.
děkan

doc. Ing. Milan Kolář, CSc.
vedoucí ústavu

V Liberci dne 9. října 2020

Prohlášení

Prohlašuji, že svou bakalářskou práci jsem vypracoval samostatně jako původní dílo s použitím uvedené literatury a na základě konzultací s vedoucím mé bakalářské práce a konzultantem.

Jsem si vědom toho, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci nezasahuje do mých autorských práv užitím mé bakalářské práce pro vnitřní potřebu Technické univerzity v Liberci.

Užiji-li bakalářskou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti Technickou univerzitu v Liberci; v tomto případě má Technická univerzita v Liberci právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Současně čestně prohlašuji, že text elektronické podoby práce vložený do IS/STAG se shoduje s textem tištěné podoby práce.

Beru na vědomí, že má bakalářská práce bude zveřejněna Technickou univerzitou v Liberci v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů.

Jsem si vědom následků, které podle zákona o vysokých školách mohou vyplývat z porušení tohoto prohlášení.

15. května 2022 Pavel Müller

Ovládání Universal Robotu z nadřízeného programu

Abstrakt

Tato bakalářská práce se zabývá problematikou vzdáleného řízeni průmyslového robota UR3 pomocí externího programu. Komunikace s robotem je založena na použití protokolu TCP/IP, který umožňuje vytvoření síťového spojení typu klient - server. Komunikace s robotem využívá klientské rozhraní reálného času. Řešení je vytvořeno v matlabu a C♯. Řešení navržené v této bakalářské práci poskytuje možnost získávat informace od robota a posílat mu příkazy v reálném čase pomocí externího systému.

Klíčová slova: Robot, vzdálené řízení, Matlab, C♯, realný čas, UR3

Universal Robot controlled by external system

Abstract

This bachelor thesis deals with the issue of remote control of an industrial robot UR3 using an external system. The communication with the robot is based on usage of TCP/IP protocol, which enables creation of client - server type network connection. Communication with the robot uses a real-time client interface. The solution itself is created in matlab and C♯. The solution provided in this bachelor's thesis provides the facility to get information from the robot and send commands to it in real time through the usage of external system.

Keywords: Robot, Remote control, Matlab, C♯, Real time, UR3

Obsah

Seznam zkratek 10

1 Úvod **11**

2 Universal robots **12**
2.1 UR3 12
2.2 Kontrolér CB 3,1 13
2.3 Teach Pendant 15

3 Způsoby řízení Universal Robot **16**
3.1 Polyscope 16
3.2 URscript 17
3.3 Vzdálené řízení přes TCP/IP 17
3.3.1 Primární a sekundární rozhraní 18
3.3.2 Real-time rozhraní 18
3.3.3 RTDE rozhraní 18

4 Výběr komunikačního rozhraní a programovacího prostředí **19**
4.1 Real-time 19
4.2 URsim 20
4.2.1 Instalace 20

5 Ovládání robota pomocí externího programu **23**
5.1 Připojení k robotovi 23
5.1.1 Připojení k robotovi pomocí C♯ 24
5.1.2 Připojení k robotovi pomocí Matlabu 25
5.2 Získávání informací 25
5.2.1 Natočení kloubů 26
5.2.2 Teploty motorů 29
5.2.3 Stav digitálních výstupů 32
5.3 Posílání příkazů 34
5.3.1 Pohybj 35
5.3.2 Pohybl 37
5.3.3 Speedj 39

6 Ukázková úloha **41**
6.1 Použití programů 41
6.2 Popis ukázkové úlohy 41
6.3 Řešení 42

7 Závěr **45**

Použitá literatura **46**

Přílohy **47**
A-CD 47

Seznam obrázků

2.1 UR3 [2] . . . 12
2.2 Kontrolér CB 3,1 [4] . . . 14
2.3 Terminál vstupů a výstupů kontroléru CB 3,1 [5] . . . 14
2.4 Prostředí polyscope . . . 15

3.1 Programovací prostředí Polyscope . . . 16
3.2 Příklad použití URscriptu . . . 17
3.3 Princip komunikace s externím zařízením . . . 18

4.1 VirtualBox úvodní obrazovka . . . 21
4.2 VirtualBox vytvoření nového virtuálního počítače . . . 21
4.3 VirtualBox výběr disku s URsim . . . 22
4.4 Úvodní obrazovka URsim . . . 22

5.1 URsim nastavení IP . . . 23
5.2 Nastavení síťového adaptéru pro komunikaci s virtualboxem . . . 24
5.3 C♯ připojení . . . 25
5.4 Matlab připojení . . . 25
5.5 C♯ informace . . . 25
5.6 Matlab informace . . . 26
5.7 C♯ natočení kloubů . . . 27
5.8 C♯ natočení kloubů použití . . . 27
5.9 Natočení kloubů příklad . . . 28
5.10 Matlab natočení kloubů . . . 29
5.11 Matlab natočení kloubů použití . . . 29
5.12 C♯ teploty motorů . . . 30
5.13 C♯ teploty motorů použití . . . 30
5.14 Teploty motorů příklad . . . 31
5.15 Matlab teploty motorů . . . 31
5.16 Matlab teploty motorů použití . . . 32
5.17 C♯ hodnota digitálních výstupů . . . 32
5.18 C♯ hodnota digitálních výstupů použití . . . 33
5.19 C♯ hodnota digitálních výstupů příklad . . . 33
5.20 URsim hodnota digitálních výstupů příklad . . . 33
5.21 Matlab hodnota digitálních výstupů . . . 34
5.22 Matlab hodnota digitálních výstupů použití . . . 34

5.23 C♯ posílání příkazů 34
5.24 Matlab posílání příkazů 35
5.25 C♯ pohybj 36
5.26 C♯ pohybj použití 36
5.27 Matlab pohybj 37
5.28 Matlab pohybj použití 37
5.29 C♯ pohybl 38
5.30 C♯ pohybl použití 38
5.31 Matlab pohybl 39
5.32 Matlab pohybl použití 39
5.33 C♯ speedj 40
5.34 C♯ speedj použití 40

6.1 Zjednodušené schéma úlohy 42
6.2 Ukázka výsledku 43
6.3 Dynamická alokace v C♯ 43
6.4 Zdrojový kód pro řešení úlohy v matlabu 44

Seznam zkratek

TUL	Technická univerzita v Liberci
FM	Fakulta mechatroniky, informatiky a mezioborových studií Technické univerzity v Liberci
Gui	Graphical user interface - Grafické uživatelské rozhraní
Cobot	collaborative robot - Kolabirativní robot
UR	Universal Robots
TCP	Tool center point - Středový bod nástroje
RTDE	Real time data exchange - Výměna dat v reálném čase

1 Úvod

Podle definice robot je „universálně použitelný pohyblivý automat s větším počtem os, jehož pohyby jsou co do svého sledu a pohybových drah, respektive úhlů volně programovatelné (tj. bez mechanického zásahu) a případně řízené senzory. Roboty mohou být vybaveny chapadly (rameny), nástroji a jinými výrobními prostředky a mohou provádět manipulační, respektive výrobní úkoly“[1]. Uplatnění robota je možné najít v oblastech teoretických, experimentálních a průmyslových.

Motivací pro tuto bakalářskou práci a jejím cílem je napsat program, který umožní ovládat právě takového průmyslového robota. Jedná se od robota UR3 od společnosti Universal Robots. K vytvoření tohoto programu jsou použití programovací jazyky Matlab nebo C♯. Oba programy nabízejí možnost získávat informace z robota o jeho aktuálním stavu a posílat některé základní příkazy k jeho ovládání, jako jsou pohybj a pohybl. Použití a zároveň důkaz správné funkce obou programů je znázorněn na ukázkové úloze. Programy tedy umožňují vzdáleně ovládat robota a použití pokročilých funkcí nabízených matlabem a C♯.

2 Universal robots

Universal robots je společnost specializující se na výrobu univerzálních průmyslových kolaborativních robotů, neboli cobotů. Od roku 2008, kdy tato společnost představila prvního komerčně použitelného robota, vyvinula řadu produktů, mezi které patří coboti UR3, UR5, UR10 a UR16, kde číslo za zkratkou „UR“ reprezentuje nosnost jednotlivých cobotů. Roboti řady URX najdou uplatnění například v úlohách montáže, upravě povrchu, svařování a mnoha dalších. Nabídka jejich cobotů je také doprovázená řadou softwerů, naučných školení a vybavením a doplňky v certifikovanými v programu UR+.

2.1 UR3

UR3 je nejmenší robot z nabídky od Universal robots. Jedná se o šestiosý kolaborativní rameno. Robot má celkem šest kloubů, což umožňuje robotovi plynulý pohyb v prostoru. Robota lze vidět na obrázku 2.1. Jednotlivé klouby robota se nazývají ze zdola nahoru základna, rameno, loket, zápěstí 1, zápěstí 2 a zápěstí 3.

Obrázek 2.1: UR3 [2]

Z tabulky 2.1 je zřejmé, že všechny klouby se mohou otáčet v úhlu ± 360°, kromě zápěstí 3, které má neomezené otáčení. Tuto schopnost lze využít například v aplikacích, jako je šroubování. Ve stejné tabulce lze také vidět, že rychlost otáčení je u všech zápěstí je 360 °/s a u ostatních kloubů je 180°/s.

Celková hmotnost robota je 11 kg a jak je patrné z názvu robota, má maximální zatížení 3 kg. Maximální zatížení se snižuje v závislosti na vzdálenost nástroje od základny robota. Dosah robota je 500 mm od základny robota. Opakovatelnost pozice je 0,1 mm, což znamená, že robot může opakovat pohyb s přesností 0,1 mm.

Veškeré technické specifikace jsou převzaty z manuálu UR3 [3]

Tabulka 2.1: Parametry otáčení a rychlostí kloubů

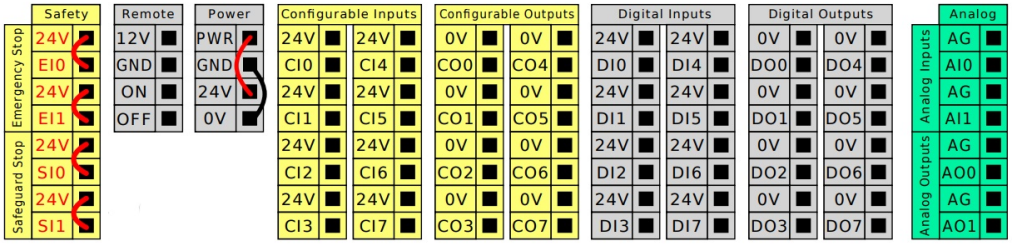
Název kloubů	Úhel otáčení	Rychlost otáčení
Základna	± 360 °	180 °/s
Rameno	± 360 °	180 °/s
Loket	± 360 °	180 °/s
Zápěstí 1	± 360 °	360 °/s
Zápěstí 2	± 360 °	360 °/s
Zápěstí 3	∞	360 °/s

2.2 Kontrolér CB 3,1

Kontrolér CB 3,1 je řídicí jednotka zodpovědná za zpracování informací, ovládání a napájení robota a komunikaci s externími zařízeními. Na obrázku 2.2 je patrné, že kontrolér je poměrně malý elektrický rozvaděč se závěsem na teach pendant. Uvnitř kontroléru se nachází CPU, napájecí zdroj a různé vstupy a výstupy. Také se zde nachází konektory pro komunikaci s externími zařízeními, jako je ethernetový port, usb 1, usb 2 a mini displayport.

Obrázek 2.2: Kontrolér CB 3,1 [4]

Jak již bylo zmíněno, kotrolér obsahuje řadu svorkovnic. Na obrázku 2.3 je možné tyto svorkovnice vidět. Žluté svorkovnice s červenými písmeny jsou bezpečnostní svorkovnice. Jsou rozděleny na dvě části. Nejprve je horní část určena pro nouzové zastavení. Zde je možné připojit například ovladač nouzového zastavení. Spodní část je určena pro bezpečnostní zastavení. Zde je možné připojit například světelné závory. Žluté svorkovnice s černými písmeny jsou konfigurovatelné vstupy a výstupy. Role těchto vstupů a výstupů může být definována v konfiguraci robota. Lze je použít například jako reset tlačítko pro ochranné zastavení. Šedé svorkovnice s černými písmeny na pravé straně jsou standardními vstupy a výstupy. Zde je možné odesílat nebo přijímat 24V signály z externích zařízení, jako jsou PLC a elektropneumatické ventily. Šedé svorkovnice s černými písmeny na levé straně slouží jako zdroj 24V. Zelené svorkovnice jsou analogové vstupy a výstupy. Ke každé jsou dvě svorkovnice. Mohou být použity například pro příjem signálu z teplotního čidla nebo pro řízení otáček motoru.



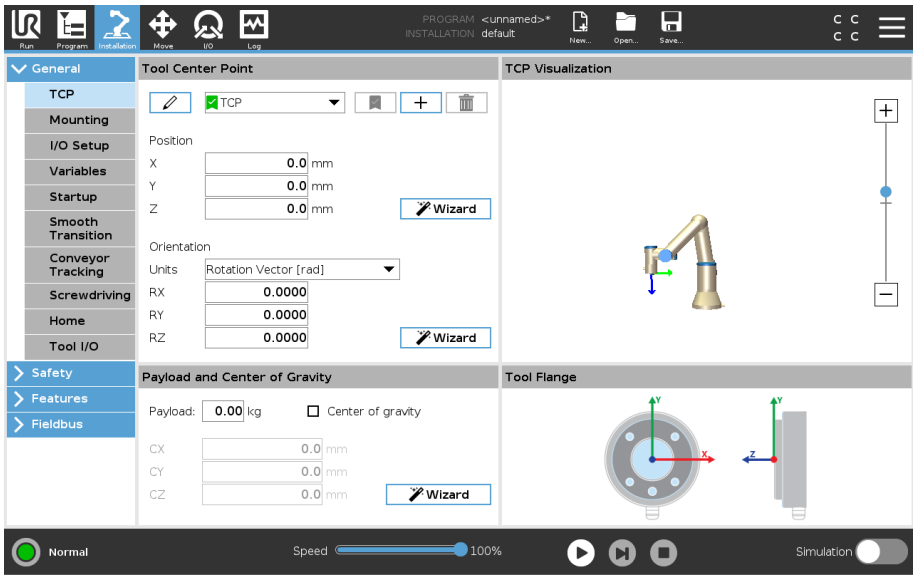
Obrázek 2.3: Terminál vstupů a výstupů kontroléru CB 3,1 [5]

2.3 Teach Pendant

Teach pendant je monitor a dotyková obrazovka pro robota. Lze jej použít k nastavení různých parametrů a ovládání robota. Uživatelské rozhraní na teach pendantu se nazývá Polyscope. Po spuštění teach pendantu se objeví zpráva s dotazem, zda chce operátor robota inicializovat. Přijetím této zprávy se obrazovka změní na obrazovku inicializace robota, kde je možné robota spustit. Po spuštění robota může operátor přejít na jednu ze šesti záložek, kterými jsou „run“, „program“„installation“ „move “ „I/O“ „Log“ .

Záložka Run poskytuje možnost načíst program ze složky. Záložka programu bude popsána v oddílu Polyscope3.1. Záložka instalace poskytuje možnost nastavení různých parametrů, jako je poloha tcp, montážní orientace robota, názvy vstupů a výstupů, bezpečnostní nastavení, jako jsou limity robota a limity kloubů a další parametry. Záložka přesun poskytuje informace o aktuální poloze tcp a natočení kloubů. Poskytuje také možnost pohybovat robotem a nastavit jej do režimu volné jízdy. Záložka I/O zobrazuje všechny vstupy a výstupy, které byly popsány v předchozím oddílu. Záložka log poskytuje informace o teplotě robota a odběru proudu a napětí.

Teach pendant také umožňuje upravit systémové nastavení, jako je jazyk, nastavení sítě, nastavení hesla a také dává možnost aktualizovat software robota.



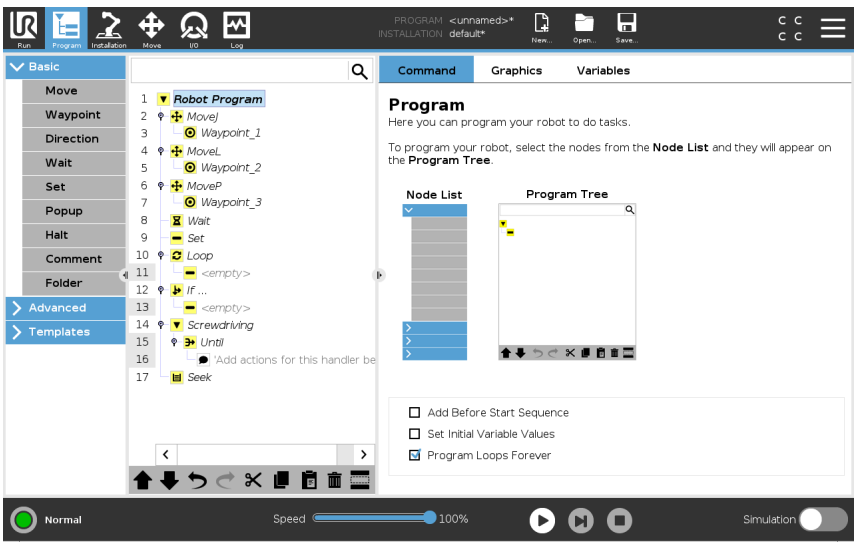
Obrázek 2.4: Prostředí polyscope

3 Způsoby řízení Universal Robot

Universal Robots nabízejí více možností, jak jejich robota ovládat. Robota lze ovládat pomocí grafického uživatelského rozhraní zvaného polyscope, skriptovacího jazyka zvaného URscript nebo pomocí dálkového ovládání přes modbus, ethernet/IP a profinet.

3.1 Polyscope

Jak již bylo zmíněno, polyscope je grafické uživatelské rozhraní zkráceně gui určené k programování, řízení robota a nastavení různých parametrů. Programovací prostředí polyscope lze vidět na obrázku 3.1. Na levé straně obrázku je výběr programovacích nástrojů. Tento výběr je rozdělen do tří skupin, a to základní, pokročilé a šablony. V základní skupině jsou funkce jako `move, waypoint, wait, set` a další. Ve skupině pokročilých funkcí jsou příkazy pro `loop, if, šroubovací funkce`, možnost zadávání příkazů URscript a další. Ve skupině šablon jsou funkce pro vyhledávání, příkaz síly, paletizaci a sledování dopravníku. Uprostřed je strom programu, který zobrazuje celý program. Program se pohybuje shora dolů. Na pravé straně se po výběru požadovaného příkazu zobrazí informace o vybraném příkazu. Je zde také možné nastavit parametry pro vybraný příkaz.



Obrázek 3.1: Programovací prostředí Polyscope

3.2 URscript

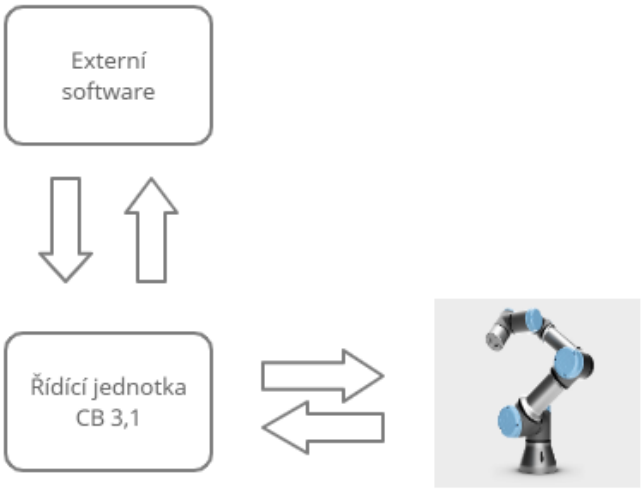
URscript je skriptovací programovací jazyk vyvinutý společností Universal Robots [6]. URscrip zahrnuje typy, proměnné a řídicí příkazy, jako jsou funkce `if` nebo `while`. Typy proměnných jsou `bool, int, float, string, pose`. Způsob programování v URscriptu je podobný základním programovacím jazykům jako je python nebo C. URscript lze napsat v libovolném textovém editoru a poté jej uložit do kontroléru a načíst v touch pendantu, pokud je zachována syntaxe URscriptu. Příklad použiti URscriptu lze vidět na obrázku 3.2.

```
if a > 4:
a = a + 1
elif b < 8:
b = b * a
else:
a = a + b
end

p = [1,2,3,4,5]
i = 0
while i < 5:
l[i] = l[i]*3
end

def add(a, b):
return a+b
end

result = add(2, 5)


movej([0,-1.57,0,-3.14,1.57,1.57],
       a=1.1, v=1.04, t=0, r=0)
```

Obrázek 3.2: Příklad použití URscriptu

3.3 Vzdálené řízení přes TCP/IP

Protokol TCP/IP je jedním ze způsobů, jak může robot UR komunikovat s externími zařízeními. Data, jako jsou informace o stavu robota nebo příkazy zasílané robotovi, lze přenášet prostřednictvím soketové komunikace. V soketové komunikaci se robot chová jako klient a kontrolér funguje jako server. Univerzální roboti nabízejí celkem čtyři porty pro komunikaci přes TCP/IP a to primární port, sekundární port, real time port a RTDE port. Princip komunikace je vidět na obrázku 3.3.



Obrázek 3.3: Princip komunikace s externím zařízením

3.3.1 Primární a sekundární rozhraní

UR kontrolér poskytuje servery pro odesílání údajů o stavu robota a přijímání příkazů URscript. Primární rozhraní lze použít k přenosu informací o stavu robota a dodatečných zpráv. Sekundární rozhraní lze použít pouze k přenosu informací o stavu robota. Obě rozhraní přijímají příkazy s přenosovou frekvencí 10 Hz. Číslo portu primárního rozhraní je 30001 a sekundárního je 30002. Rozhraní 30011 a 30012 lze použít v aplikacích, kde je potřeba pouze monitorovat stav robota.

3.3.2 Real-time rozhraní

Rozhraní Real-time funguje podobně jako primární a sekundární porty. Může být použit k přenosu informací o stavu robota a odesílání příkazů robotu. Velký rozdíl oproti dvěma výše zmíněným rozhraní je přenosová rychlost.Rrozhraní Real-time může přenášet data frekvencí 125 Hz. Číslo portu rozhraní Real-time je 30003. Rozhraní 30013 lze použít v aplikacích, kde je pouze potřeba ke sledování stavu robota.

3.3.3 RTDE rozhraní

Rozhraní RTDE neboli rozhraní pro výměnu dat v reálném čase poskytuje způsob, jak synchronizovat externí aplikace s kontrolérem UR, aniž by došlo k porušení jakýchkoli vlastností kontroléru UR v reálném čase. Rozhraní RTDE může přenášet data s frekvencí 125 Hz. Číslo portu rozhraní RTDE je 30004.

4 Výběr komunikačního rozhraní a programovacího prostředí

Jak bylo zmíněno v předchozí kapitole Universal Robots nabízí čtyři klientská rozhraní pro dálkové ovládání robota. Protože požadavkem této bakalářské práce je řídit robota v reálném čase, jsou na výběr pouze dvě možnosti a to rozhraní Real-time a RTDE. Z těchto dvou bylo vybráno rozhraní v reálném čase. Programovací rozhraní zvolená pro programování robota jsou matlab a C♯. Většina řešení byla provedena v URsim a poté testována a upravena na skutečného robota.

4.1 Real-time

Komunikační rozhraní Real-time posílá veškeré informace o robotovi jako jednu dlouhou zprávu. Délka této zprávy se může lišit v závislosti na verzi Real-time na robotu. Pro Real-time verzi 5,9 je tato zpráva dlouhá 1116 bajtů a pro Real-time verzi 5,10 1220 bajtů. Tato zpráva může být aktualizována a odeslána každých 8 ms.

Získání informací o stavu jednotlivých robotů z této zprávy lze provést rozdělením zprávy na více částí. Tabulka 4.1 ukazuje, jak najít informace ve zprávě. Pokud se například celá zpráva převede z typu `bytes` na typ `double`, pak informace o úhlovém natočení kloubů lze nalézt mezi řádky 32 a 37. Níže uvedená tabulka neukazuje všechny stavy robota, které lze zjistit[7].

Tabulka 4.1: Informace o aktuálním stavu robota poskytované přes port 30003

Význam	Datový typ	Počet hodnot	Velikost v bajtech	Pozice řádku
Délka zprávy	Integer	1	4	
Cílový moment	Double	6	64	26-31
Natočení kloubů	Double	6	64	32-37
Rychlost kloubů	Double	6	64	38-43
Zrychlení kloubů	Double	6	64	44-49
Proud kloubů	Double	6	64	50-55
Poloha tcp	Double	6	64	56-61
Rychlost tcp	Double	6	64	62-67
Síla tcp	Double	6	64	68-73
Digitální vstupy	Double	1	8	86
Tepolota motorů	Double	6	64	87-92
Digitální výstupy	Double	1	8	131

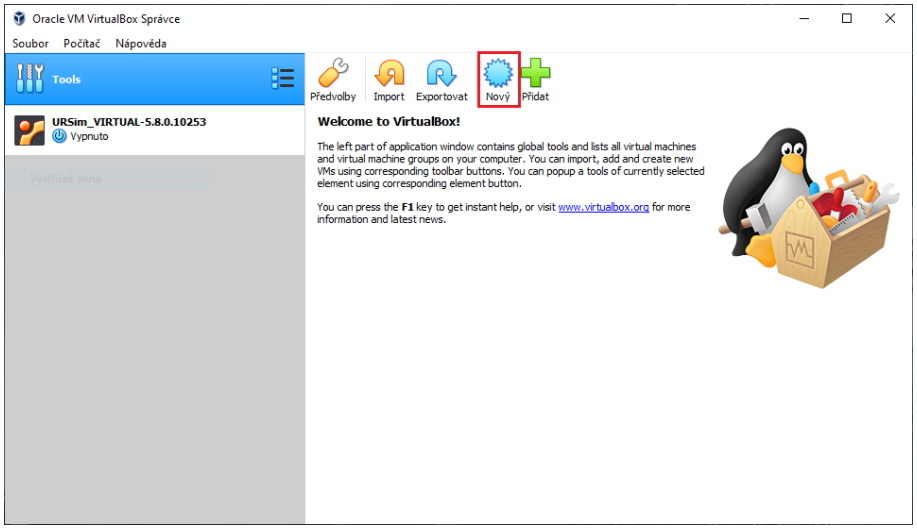
4.2 URsim

Ursim je freeware, který lze stáhnout z webu Universal Robots[8]. Ursim lze použít k programování robota offline. Simulátor umožňuje realistickou simulaci robota, což znamená, že simulovaný robot se chová a pohybuje stejným způsobem jako skutečný robot. Rozhraní Ursim je úplně stejné jako rozhraní Polyscope v Teach pendantu. To umožňuje snadný přechod z programování robota přes Teach pendant do programování v URsim. Programy v URsim lze uložit a nahrát do robota.

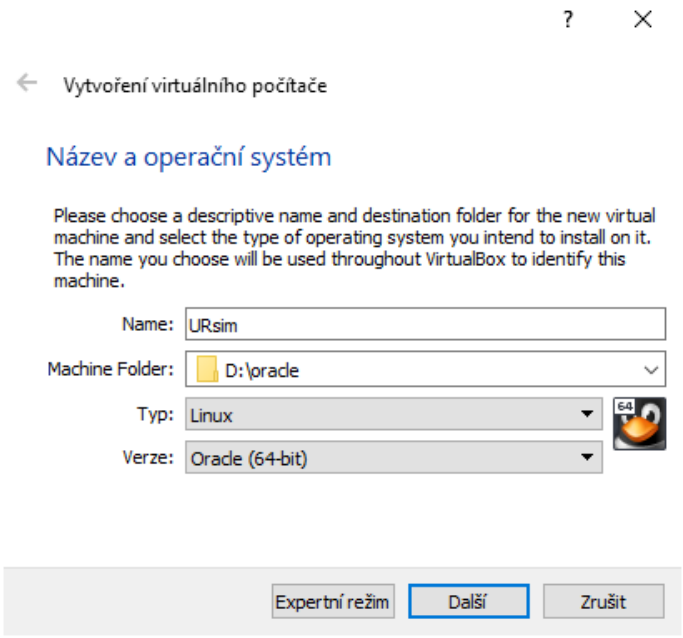
4.2.1 Instalace

Protože URsim funguje pouze na operačním systému linux, je nutné buď mít počítač s linuxem, nebo vytvořit virtuální počítač s linuxem. V této bakalářské práci byla použita druhá možnost. K vytvoření virtuálního počítače byl použit program `Oracle VM VirtualBox`[9].

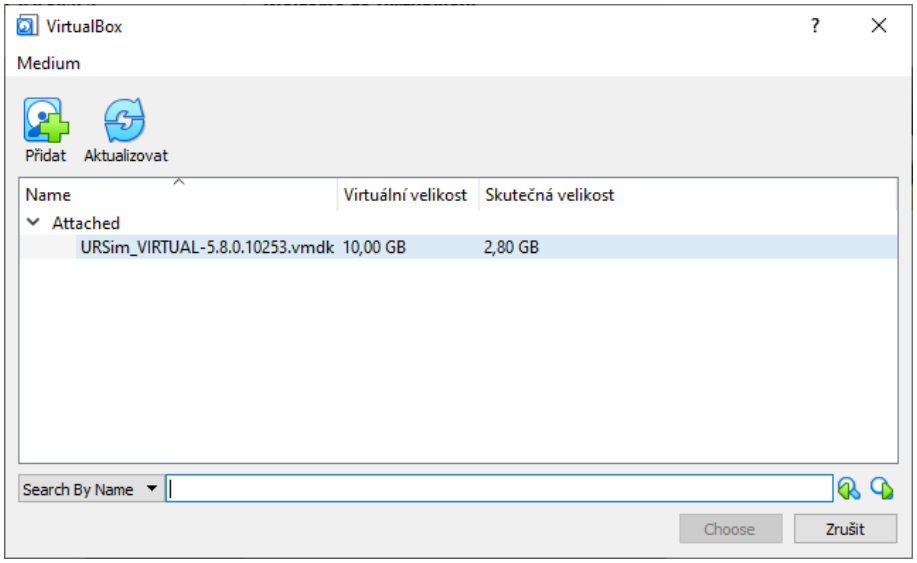
Po otevření programu se zobrazí obrazovka viditelná na obrázku 4.1. Kliknutím na volbu „Nový“ zvýrazněnou v červeném čtverci inicializujete vytváření nového virtuálního počítače a otevře se okno viditelné na 4.2. V tomto okně lze vybrat některé počáteční parametry, jako je název virtuálního počítače, jeho umístění na disku, typ systému a jeho verze. Na další obrazovce lze zvolit velikost RAM virtuálního počítače. Na poslední obrazovce je vybrán pevný disk. Zde volba „Použít existující soubor s virtuálním pevným diskem“ umožňuje použití obrazu virtuálního disku URsim [8], jak je znázorněno na obrázku 4.3. Poslední obrázek 4.4 ukazuje prostředí URsim po spuštění virtuálního počítače. Zde lze vybrat URsim pro každého ze čtyř univerzálních robotů. V tomto případě se použije URsim pro UR3. Po otevření URsim se zobrazí stejné uživatelské rozhraní jako na teach pendantu.



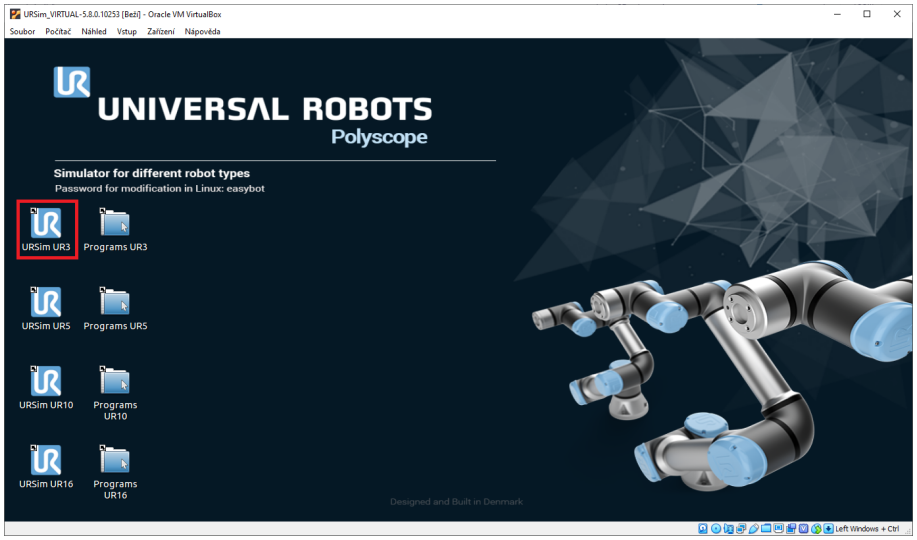
Obrázek 4.1: VirtualBox úvodní obrazovka



Obrázek 4.2: VirtualBox vytvoření nového virtuálního počítače



Obrázek 4.3: VirtualBox výběr disku s URsim



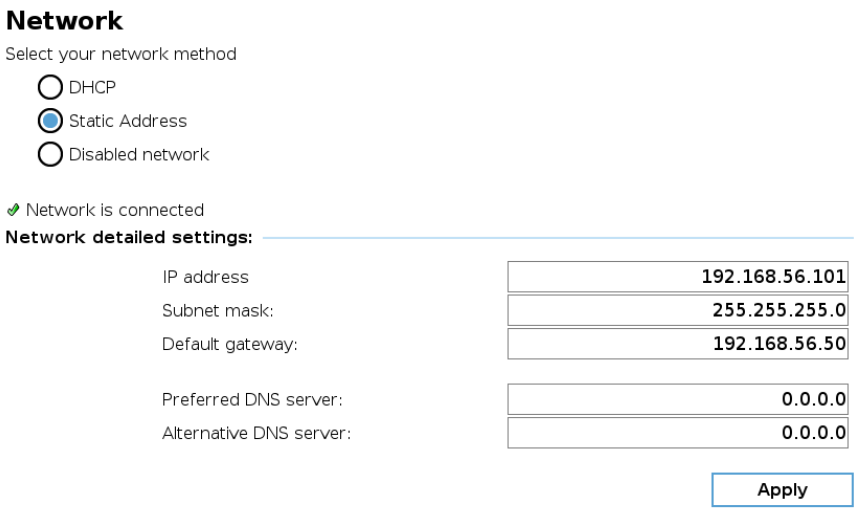
Obrázek 4.4: Úvodní obrazovka URsim

5 Ovládání robota pomocí externího programu

Ovládání robota se skládá ze tří částí a to připojení k robotu, přijímání informací o stavu robota a odesílání příkazů robotu. Tento oddíl se zabývá těmito třemi částmi. Na začátku je vysvětleno a ukázáno, jak se připojit k robotu. V druhé části je vysvětleno, jak získat informace z robota a ukázáno na příkladech zjišťování natočení kloubů, teploty motorů a stavu digitálních výstupů. V poslední části je vysvětleno, jak posílat robotovi příkazy a ukázáno na příkladech pohybových příkazů pohybj, pohybl a speedj. Vše je vysvětleno a ukázáno pro Matlab a C♯.

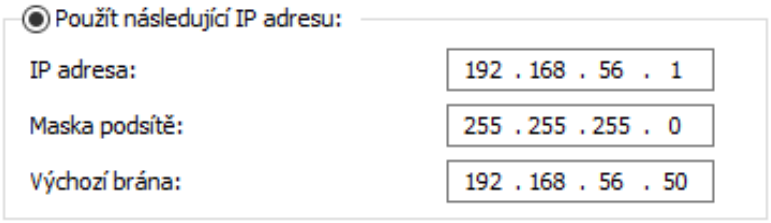
5.1 Připojení k robotovi

Předtím než je možné se k robotovi vůbec připojit, tak je nutné správně nastavit IP adresy robota a počítače, ze kterého probíhá komunikace s robotem. Pro zjištění síťového nastavení robota stačí v PolyScope přejít do nabídky „Settings“, „System“ a „Network“. V tuto chvíli se zobrazí síťové nastavení robota, viz obrázek 5.1. Zde lze vidět IP adresa, kterou má robot přidělenou. Pokud žádnou adresu přidělenou nemá, tak je nutné ji nastavit.



Obrázek 5.1: URsim nastavení IP

Dalším krokem je nastavení počítače do stejné sítě podle výše uvedeného obrázku 5.1. Pro správné nastavení je nutné, aby první tři bajty IP adresy na počítači byli stejné jako na robotovi. Dále je nutné, aby maska podsítě a výchozí brána byli stejné. Například pokud na robotovi je nastavená IP adresa 192.168.56.101, maska podsítě 255.255.255.0 a výchozí brána 192.168.56.50, tak na počítači musí být nastavená IP adresa 192.168.56.x, maska podsítě 255.255.255.0 a výchozí brána 192.168.56.50. Možné nastavení síťového připojení na počítači je uvedeno na obrázku 5.2. Pro ověření správného nastavení je dobré zkontrolovat jestli robot a počítač jsou schopný spolu komunikovat. To lze provést jednoduše otevřením příkazového řádku na počítači a posláním příkazu `Ping`. Pro robota nastaveného jako je na obrázku 5.1 by tento příkaz vypadal následovně `Ping 192.168.56.101`.



Obrázek 5.2: Nastavení síťového adaptéru pro komunikaci s virtualboxem

5.1.1 Připojení k robotovi pomocí C♯

Pro připojení k robotovi pomocí C♯ je nutné použít knihovnu `Net.Sockets`, Ta se linkuje příkazem `using System.Net.Sockets;`. Kód pro připojení k robotovi lze vidět na obrázku 5.3. Příkazem `TcpClient client = new TcpClient();` se vytvoří nový tcp klient pod názvem „client“. Příkazem `client.Connect(server, port);` vznikne propojení s robotovi podle IP adresy a portu. Příkazem `network_stream = client.GetStream();` se získá datový proud pro přijímání a odesílání dat a uloží se do „network_stream“. Příkazy `network_stream.Dispose();` a `client.Close();` se odstraní veškeré prostředky používané datovým proudem a uzavře se komunikace s klientem.

```
Int32 port = 30003;
string server = "192.168.56.101";
TcpClient client = new TcpClient();
if (client.Connected == false)
{

    client.Connect(server, port);
}
network_stream = client.GetStream();
network_stream.Dispose();
client.Close();
```

Obrázek 5.3: C♯ připojení

5.1.2 Připojení k robotovi pomocí Matlabu

Pro připojení k robotovi pomocí matlabu je nutné nainstalovat toolbox `Instrument Control Toolbox` [10]. Tento toolbox umožní vytvořit objekt reprezentující připojení ke vzdálenému hostu pomocí matlabu. Kód pro připojení k robotovi lze vidět na obrázku 5.4. Příkazem `client = tcpclient(ip, port);` se vytvoří objekt připojení ke vzdálenému hostu a uloží se do proměnné „client“. Vstupem do funkce `tcpclient` je IP adresa robota a vybraný port. Příkazem `clear client;` se poté tato komunikace uzavře.

```
ip = '192.168.56.101';
port = 30003;
client = tcpclient(ip, port);
clear client
```

Obrázek 5.4: Matlab připojení

5.2 Získávání informací

Na obrázcích 5.5 a 5.6 je vidět způsob získávání informací v jednotlivých programech.

```
byte[] packet = new byte[1116];
network_stream.Read(packet, 0, packet.Length);
```

Obrázek 5.5: C♯ informace

```
response = read(client,funkce.bytes);
```

Obrázek 5.6: Matlab informace

Jak už bylo zmíněno v oddílu 4.1, RealTime klient posíla veškeré informace o aktuálním stavu robota jako jednu zprávu dlouhou 1116 anebo 1220 bajtů. Dále v tabulce 4.1 v oddílu 4.1 je znázorněno umístění jednotlivých informací o robotovi ve přijaté zprávě. To znamená, že pro získání informace o aktuální pozici středového bodu nástroje neboli tcp, tak z přijaté zprávy je nutné vybrat pouze řádky 56-61, kde řádek 56 reprezentuje první hodnotu polohy, tedy vzdálenost tcp na ose x od základny robota a řádek 61 reprezentuje poslední hodnotu polohy, tedy natočení tcp v ose z vůči bázi robota. Pomocí této metodiky jsou dále tvořené programy v C♯ a v matlabu. Tyto programy umožní uživateli získat z robota právě ty informace, které požaduje. V dalším oddílu jsou uvedeny příklady použití těchto programů.

5.2.1 Natočení kloubů

Informace natočení kloubů robota je velmi důležitou informací s pohledu ovládání robota. Tato informace udává aktuální natočení všech šesti kloubů robota. Tato informace se využívá zejména při ovládání robota v prostoru kloubů. Níže jsou uvedeny kódy pro získání této informace v C♯ a v matlabu a použití těchto funkcí.

C♯

Na obrázku 5.7 lze vidět zdrojový kód metody pro získání informace o aktuálním natočení kloubů. Jméno této metody je „kloub_natoceni_skut“. Očekávaným výstupem této metody je jedno-dimenzionální pole s šesti hodnotami datového typu double. Proto je nutné metodu definovat jako `public double[]`, kde „public“ specifikuje úroveň přístupu a „double[]“ definuje typ hodnoty, kterou funkce vrátí. V samotné metodě je nejdříve nutné definovat proměnnou, do které se budou ukládat hodnoty. To je provedeno příkazem `double[] vystup = new doulbe[6]`. Dále je potřeba zkontrolovat zda už je vytvořené připojení k robotovi, a jestliže není, tak se vytvoří. Jak již bylo zmíněno na začátku oddílu 5.2, příkaz `network_stream.Read(packet,0,packet.length` uloží informace z robota do pole „packet“, které je následovně obrácené. V tuto chvíli jsou všechny informace o roboto uložené v datovém typu byte, takže je nutné je konvertovat do typu double. Toho je docíleno použitím metody `BitConverter.ToDouble`, která vezme osm bajtů z pole „packet“ na startovní pozici určené podle tabulky 4.1 a konvertuje je do datového typu double. Nakonec je tato hodnota vynásobená konstantou `180/PI`, která zaručí převod radiánů na stupně.

```
0 references
public double[] kloub_natoceni_skut(Int32 port, string server)
{

    double[] vystup = new double[6];
    if (client.Connected == false)
    {
        client.Connect(server, port);
    }


    network_stream = client.GetStream();
    network_stream.Read(packet, 0, packet.Length);
    Array.Reverse(packet);

    vystup[0] = BitConverter.ToDouble(packet, packet.Length - prvni_packet - ((32 ) * 8 )) * 180 / Math.PI;
    vystup[1] = BitConverter.ToDouble(packet, packet.Length - prvni_packet - ((33 ) * 8 )) * 180 / Math.PI;
    vystup[2] = BitConverter.ToDouble(packet, packet.Length - prvni_packet - ((34 ) * 8 )) * 180 / Math.PI;
    vystup[3] = BitConverter.ToDouble(packet, packet.Length - prvni_packet - ((35 ) * 8 )) * 180 / Math.PI;
    vystup[4] = BitConverter.ToDouble(packet, packet.Length - prvni_packet - ((36 ) * 8 )) * 180 / Math.PI;
    vystup[5] = BitConverter.ToDouble(packet, packet.Length - prvni_packet - ((37 ) * 8 )) * 180 / Math.PI;
    return vystup;
}
```

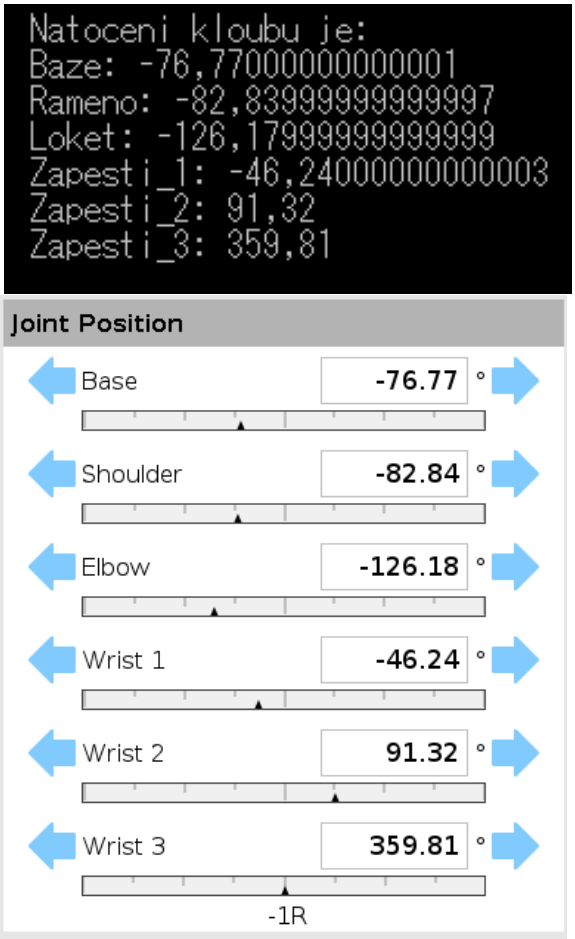
Obrázek 5.7: C♯ natočení kloubů

Použití metody je znázorněné na obrázku 5.8. Nejdříve je nutné definovat proměnnou, do které se informace uloží. Poté už stačí použít předem popsanou metodu.

```
double[] kloub_natoceni_skut;
kloub_natoceni_skut = robot.kloub_natoceni_skut(port, server);
```

Obrázek 5.8: C♯ natočení kloubů použití

Na obrázku 5.9 je znázorněn možný výstup metody „kloub_natoceni_skut“, kde na pravém obrázku je znázorněné aktuální natočení kloubů robota v URsim a na levém obrázku jsou hodnoty získané použitím metody.



Obrázek 5.9: Natočení kloubů příklad

Matlab

Způsob získání informace o aktuálním natočení kloubů robota je v matlabu poměrně jednodušší než v C♯. Jednak zde není potřeba definovat datový typ výstupu funkce a proměnnou, do které se bude informace ukládat. Na obrázku 5.10 lze vidět kód funkce v matlabu. Do proměnné „response“ se uloží informace přijaté od robota. Proměnná „offset“ definuje první bajt požadované informace a proměnná „count“ definuje přesnou pozici požadované informace v jednotlivých iterací funkce `for`. Do proměnné „item“ se uloží osm bajtů z proměnné „response“, které jsou poté konvertovány do datového typu double funkcí `typecast` a převedeny do stupňů vynásobením konstanty `180/PI`.

```
function  output = kloub_natoceni_skut(client);
offset = 30;
response = flip(read(client,bytes));
   for k =1:1:6
   count = length(response) - 4 -((k+offset)*8);
   item = response(count-7:count);
   output(k) = typecast(uint8(item),'double')*180/pi;
   end
end
```

Obrázek 5.10: Matlab natočení kloubů

Použití funkce je znázorněné na obrázku 5.11. Stačí použít předem popsanou funkci v programu. Funkcí `disp()` je možné zobrazit přijatou hodnotu.

```
kloub_natoceni_skut = UR3_.kloub_natoceni_skut(client);
disp(kloub_natoceni_skut);
```

Obrázek 5.11: Matlab natočení kloubů použití

5.2.2 Teploty motorů

Další užitečnou informací je teplota motorů v jednotlivých kloubech. Významnost této informace spočívá zejména z hlediska bezpečnosti. Při běžném provozu by se měla teplota robota pohybovat okolo 40°C. Pokud informace o teplotě přijata z robota ukazuje, že je teplota jednoho nebo více kloubů větší než 50°C, tak je dobré robota zkontrolovat pro možnou chybu. Níže jsou uvedeny kódy pro získání teploty kloubů v C a v matlabu a použití těchto funkcí.

C♯

Na obrázku 5.12 lze vidět zdrojový kód metody pro získání informace o aktuální teplotě kloubů. Jméno této metody je „motor_teplota“. Princip funkce této metody je úplně stejný jako u výše uvedené metody pro získání natočení kloubů. Jediným rozdílem je pozice požadované informace v poli „packet“. Zde se tyto informace nacházejí na pozicích 87-92. Protože motor posílá informaci o teplotě ve stupních Celsia, tak zde není nutné provádět přepočet.

```
0 references
public double[] motor_teplota(Int32 port, string server)
{

    double[] vystup = new double[6];
    if (client.Connected == false)
    {
        client.Connect(server, port);
    }
    network_stream = client.GetStream();
    network_stream.Read(packet, 0, packet.Length);
    Array.Reverse(packet);

    vystup[0] = BitConverter.ToDouble(packet, packet.Length - prvni_packet - ((87 ) * 8 ));
    vystup[1] = BitConverter.ToDouble(packet, packet.Length - prvni_packet - ((88 ) * 8 ));
    vystup[2] = BitConverter.ToDouble(packet, packet.Length - prvni_packet - ((89 ) * 8 ));
    vystup[3] = BitConverter.ToDouble(packet, packet.Length - prvni_packet - ((90 ) * 8 ));
    vystup[4] = BitConverter.ToDouble(packet, packet.Length - prvni_packet - ((91 ) * 8 ));
    vystup[5] = BitConverter.ToDouble(packet, packet.Length - prvni_packet - ((92 ) * 8 ));
    return vystup;
}
```

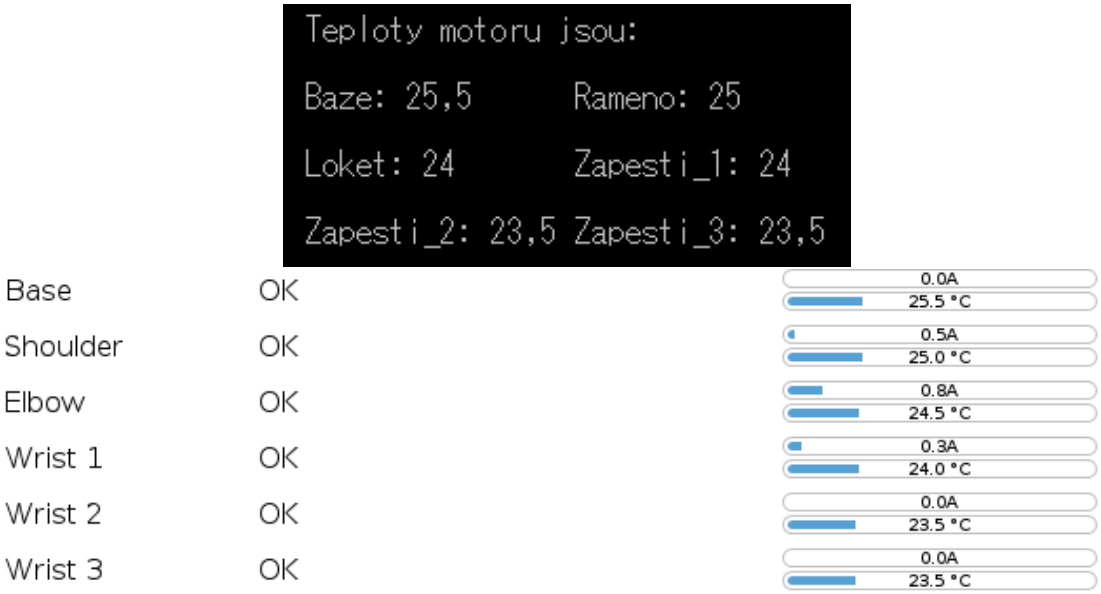
Obrázek 5.12: C♯ teploty motorů

Použití metody je znázorněné na obrázku 5.13. Nejdříve je nutné definovat proměnnou, do které se informace uloží. Poté už stačí použít předem popsanou metodu.

```
double[] motor_teplota;
motor_teplota = robot.motor_teplota(port, server);
```

Obrázek 5.13: C♯ teploty motorů použití

Na obrázku 5.14 je znázorněn možný výstup metody „motor_teplota“, kde na horním obrázku je znázorněná teplota, která byla získaná použitím metody a spodním obrázku jsou znázorněný aktuální teploty robota v URsim.



Obrázek 5.14: Teploty motorů příklad

Matlab

Na obrázku 5.15 lze vidět zdrojový kód funkce pro získání informace o aktuální teploty kloubů. Jméno funkce je „motor_teplota“. Podobně jako v předchozím případě se tato funkce nějak neliší od funkce pro získání natočení kloubů v matlabu. Opět jediným rozdílem je samotná pozice požadované informace v poli „response“, to je zajištěno proměnnou „offset“, která je zde rovná 85.

```
function  output = motor_teplota(client);
offset = 85;
response = flip(read(client,bytes));
   for k =1:1:6
   count = length(response) - 4 -((k+offset)*8);
   item = response(count-7:count);
   output(k) = typecast(uint8(item),'double');
   end

end
```

Obrázek 5.15: Matlab teploty motorů

Použití funkce je znázorněné na obrázku 5.16. Stačí použít předem popsanou funkci v programu. Funkcí `disp()` je možné zobrazit přijatou hodnotu.

```
motor_teplota = UR3_.motor_teplota(client);
disp(motor_teplota);
```

Obrázek 5.16: Matlab teploty motorů použití

5.2.3 Stav digitálních výstupů

Další užitečnou informací je stav digitálních výstupů. Tato informace říká, které digitální výstupy jsou aktivní a které ne. To může být užitečné pro kontrolu, zda výstupy robotů fungují tak, jak mají. Robot má celkem osm digitálních výstupů, osm konfigurovatelných digitálních výstupů a dva nástrojové výstupy. Každý digitální výstup má přiřazenou hodnotu v rozsahu od 2^0 pro výstup 0 až 2^7 pro výstup 7 pro běžné výstupy, 2^8 až 2^{15} pro konfigurovatelné výstupy a 2^{16} a 2^{17} pro výstupy nástroje. Návratová hodnota stavu digitálního výstupu je jedna hodnota, což znamená, že konečná hodnota je součtem všech aktivních výstupů.

C♯

Na obrázku 5.17 lze vidět zdrojový kód metody pro získání informace o aktuálním stavu výstupů. Jméno této metody je „dig_vystup“. Princip funkce této metody je úplně stejný jako u výše uvedené metody pro získání natočení kloubů. Jediným rozdílem je pozice požadované informace v poli „packet“. Protože stav digitálních výstupů je pouze jedna hodnota, tak není nutné použít pole a stačí pouze proměnnou double. Tato informace se nachází na pozici 131.

```
1 reference
public double dig_vystup(Int32 port, string server)
{
TcpClient client = new TcpClient();

double vystup ;
    if (client.Connected == false)
    {
        client.Connect(server, port);
    }
    network_stream = client.GetStream();
    network_stream.Read(packet, 0, packet.Length);
    Array.Reverse(packet);

    vystup = BitConverter.ToDouble(packet, packet.Length - prvni_packet - ((131 ) * 8 ));
    network_stream.Dispose();
    client.Close();
    return vystup;
}
```

Obrázek 5.17: C♯ hodnota digitálních výstupů

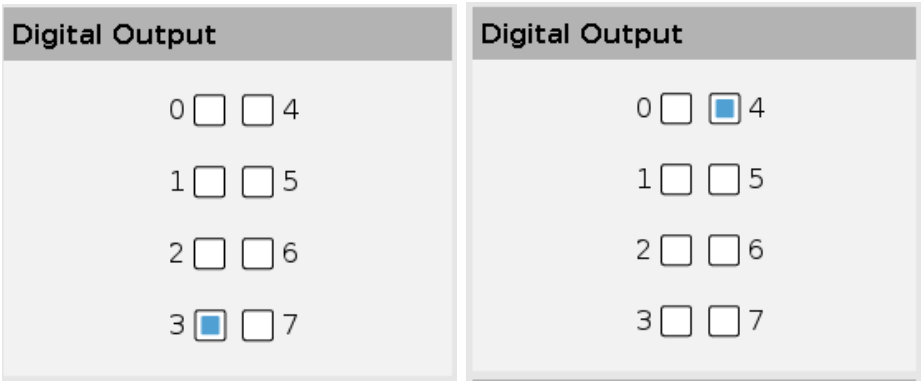
Použití metody je znázorněné na obrázku 5.18. Nejdříve je nutné definovat proměnnou, do které se informace uloží. Poté už stačí použít předem popsanou metodu.

```
double dig_vystup;
dig_vystup = robot.dig_vystup(port, server);
```

Obrázek 5.18: C♯ hodnota digitálních výstupů použití

Na obrázku 5.19 jsou znázornění možný výstupy metody „digvystup“. obrázek 5.20 ukazuje nastavení digitálních výstupů během příkladu.

Obrázek 5.19: C♯ hodnota digitálních výstupů příklad



Obrázek 5.20: URsim hodnota digitálních výstupů příklad

Matlab

Na obrázku 5.21 lze vidět zdrojový kód funkce pro získání informace o aktuálním stavu digitálních výstupů. Jméno funkce je „dig_vystup“. Podobně jako v předchozím případě se tato funkce nějak neliší od funkce pro získání natočení kloubů v matlabu. Opět jediným rozdílem je samotná pozice požadované informace v poli „response“, to je zajištěno proměnnou „offset“, která je zde rovná 130.

```
function  output = dig_vystup(client);
   offset = 130;
   response = flip(read(client,bytes));

   count = length(response) - 4 -((offset)*8);
   item = response(count-7:count);
   output = typecast(uint8(item),'double');

end
```

Obrázek 5.21: Matlab hodnota digitálních výstupů

Použití funkce je znázorněné na obrázku 5.22. Stačí použít předem popsanou funkci v programu. Funkcí `disp()` je možné zobrazit přijatou hodnotu.

```
dig_vystup = UR3_.dig_vystup(client);
disp(dig_vystup);
```

Obrázek 5.22: Matlab hodnota digitálních výstupů použití

5.3 Posílání příkazů

Tato část je věnována způsobu, jak posílat příkazy robotu. Kontrolér je schopen přijímat příkazy URscriptu, proto je nejdůležitější dávat pozor na dodržení syntaxe URscriptu. Je také nutné posílat příkazy v datovém typu string, jinak je kontrolér nerozpozná a buď se na teach pendantu objeví zpráva o chybným zadání příkazu, nebo nebude vůbec reagovat. V rámci příkazů URscript existuje také řada příkazů, které vracejí informace o stavu robota, jako je `get_actual_joint_positions()`. Při pokusu o použití některého z těchto příkazů však robot odešle zpět celou zprávu, jak je popsáno v oddílu 5.2. Proto jsou tyto příkazy v případě této bakalářské práce prakticky nepoužitelné a mohou být opomíjeny. Níže jsou popsány tři příkazy pohybj, pohybl a speedj.

Obrázek 5.23 ukazuje způsob, jak posílat příkazy robotovi přes C♯. Nejprve je příkaz zakódován do sekvence bajtů a uložen do proměnné „packet_cmd“. Poté se jednoduše odešle do robota pomocí funkce `network_stream.Write`. Vstupy funkce jsou samotný příkaz, offset a počet bajtů, ze kterých se příkaz skládá.

```
packet_cmd = utf8.GetBytes("PŘÍKAZ PODLE URSCRIPTU");
network_stream.Write(packet_cmd, 0, packet_cmd.Length);
```

Obrázek 5.23: C♯ posílání příkazů

Obrázek 5.24 ukazuje způsob, jak posílat příkazy robotu přes Matlab. Metoda je velmi podobná jaok v C♯. Příkaz se opět uloží jako řetězec do proměnné „cmd“, ale v tomto případě jako `string`. Příkaz je poté odeslán do robota pomocí funkce `writeline`. Vstupy funkce jsou připojené zařízení a příkaz.

```
cmd = ['PŘÍKAZ PODLE URSCRIPTU'];
writeline(client,cmd);
```

Obrázek 5.24: Matlab posílání příkazů

5.3.1 Pohybj

Příkaz pohybj je jedním z nejpoužívanějších pohybových příkazů robota. Po obdržení tohoto příkazu se robot přesune do určené polohy. Pohyb se provádí v lineárním kloubovém prostoru. Možné vstupy tohoto příkazu jsou póza, zrychlení, rychlost, čas, poloměr. Póza je vektor šesti hodnot, kde každá hodnota udává úhlové posunutí jednotlivých kloubů. Zrychlení a rychlost udávají zrychlení a rychlost vedoucí osy. Čas udává čas provedení pohybu, pokud je použit robot ignoruje nastavení zrychlení a rychlosti. Poloměr při použití udává poloměr zaoblení.

C♯

Obrázek 5.25 ukazuje zdrojový kód metody pro příkaz pohybj. Protože metoda nevrací žádné hodnoty, jedná se o `Public void`. Vstupy metody jsou takové, jak bylo popsáno výše, kde „Q“ je pozice, „v“ je rychlost, „a“ je zrychlení, „t“ je čas a „r“ je poloměr. Protože robot pracuje s radiány, vektor pozice se nejprve převede ze stupňů na radiány. Poté je příkaz odeslán robotovi způsobem, který je znázorněn na začátku oddílu 5.3. Protože robot očekává hodnotu ve formátu `1.234`, musí být hodnoty převedeny z číselného formátu `1,234`. To je provedené zadáním parametrů v příkazu `ToString()`.

```
2 references
public void pohybj(Int32 port, string server, double[] Q, string v, string a, string t, string r)
{
    if (client.Connected == false)
    {

        client.Connect(server, port);
    }

    for (int k = 0; k < 6; k++)
    {
        Q[k] = Q[k] * Math.PI / 180;
    }
    if (t != "0")
    {
        packet_cmd = utf8.GetBytes("movej([" +
        Q[0].ToString("G", CultureInfo.InvariantCulture) + "," +
        Q[1].ToString("G", CultureInfo.InvariantCulture) + "," +
        Q[2].ToString("G", CultureInfo.InvariantCulture) + "," +
        Q[3].ToString("G", CultureInfo.InvariantCulture) + "," +
        Q[4].ToString("G", CultureInfo.InvariantCulture) + "," +
        Q[5].ToString("G", CultureInfo.InvariantCulture) +
        "], t =" + t + ", r=" + r + ")" + "\n");
    }
    else
    {
       packet_cmd = utf8.GetBytes("movej([" +
       Q[0].ToString("G", CultureInfo.InvariantCulture) + "," +
       Q[1].ToString("G", CultureInfo.InvariantCulture) + "," +
       Q[2].ToString("G", CultureInfo.InvariantCulture) + "," +
       Q[3].ToString("G", CultureInfo.InvariantCulture) + "," +
       Q[4].ToString("G", CultureInfo.InvariantCulture) + "," +
       Q[5].ToString("G", CultureInfo.InvariantCulture) +
       "], a =" + a + ", v=" + v + ", r=" + r + ")" + "\n");
    }
    network_stream = client.GetStream();

    network_stream.Write(packet_cmd, 0, packet_cmd.Length);
}
```

Obrázek 5.25: C♯ pohybj

Na obrázku 5.26 je vidět příklad použití. V případě zobrazeném na obrázku je rychlost nastavena na 1,05 rad/s, zrychlení je 1,4 rad/s^2, čas je 0 s a rádius je 0 m. To znamená, že se robot bude otáčet rychlostí 1,04 rad/s a se zrychlením 1,4 rad/s^2. Úhlové posunutí cíleného je kloubu `[150, -30, 90, 40, 60, 70]`. To znamená, že cílový úhel pro kloub základny je 150°, pro rameno -30°, pro loket 90°, zápěstí 1 40°,zápěstí 2 60°,zápěstí 3 70°.

```
string rychlost = "1.05";
string zrychleni = "1.4";
string t = "0";
string r = "0";
double[] poloha = { 150, -30, 90, 40, 60, 70 };
robot.pohybj(port, server, poloha, rychlost, zrychleni, t, r);
```

Obrázek 5.26: C♯ pohybj použití

Matlab

Způsob odesílání příkazu robotu je podobný jako v případě C♯. Nejprve je třeba převést vstupní polohu ze stupňů na radiány. Poté je třeba čísla převést do formátu

`string`. Celý příkaz je uložen do proměnné „cmd“ a poté odeslán robotovi pomocí funkce `writeline`. Zdrojový kód v matlabu lze vidět na obrázku 5.27.

```
function pohybj(client,q,a,v,t,r)
        q = q*pi/180;

        if t > 0
        cmd =
['movej([',num2str(q(1)),',',num2str(q(2)),',',num2str(q(3)),',',num2str(q(4)),',',num2str(q(5)),',',num2str(q(6)),'],t
=',num2str(t),',r=',num2str(r),')'];
        writeline(client,cmd);
        else
        cmd =
['movej([',num2str(q(1)),',',num2str(q(2)),',',num2str(q(3)),',',num2str(q(4)),',',num2str(q(5)),',',num2str(q(6)),'],
a=',num2str(a),',v=',num2str(v),',r=',num2str(r),')'];
        writeline(client,cmd);
        end
end
```

Obrázek 5.27: Matlab pohybj

Na obrázku 5.28 je vidět příklad použití v matlabu. Vstupní parametry pro pohybj jsou stejné jako v případě pro C♯ až na rychlost a zrychlení, které jsou 1,1 rad/s a 1,4 rad/s^2.

```
rychlost = 1.1;
zrychleni = 1.4;
t = 0;
r = 0;
poloha = [150, -30, 90, 40, 60, 70];
UR3_lib.pohybj(client, poloha, zrychleni, rychlost, t, r);
```

Obrázek 5.28: Matlab pohybj použití

5.3.2 Pohybl

Příkaz pohybl je také často používaným příkazem pohybu robota. Stejně jako v předchozím případě tento příkaz přesune robota do určené polohy. Pohyb je prováděn lineárně v nástrojovém prostoru. Možné vstupy pro tuto funkci jsou stejné jako v předchozím případě, tedy pozice, zrychlení, rychlost, čas a poloměr. Všechny mají stejný význam jako v předchozím případě kromě pózy. V tomto případě příkaz pracuje v kartézském prostoru a očekává tři hodnoty vzdálenosti a tři hodnoty úhlu. Vektor pozice pak vypadá takto `[x,y,z,rx,ry,rz]`.

C♯

Obrázek 5.29 ukazuje zdrojový kód metody pro příkaz pohybl. Vstupy metody jsou jak je uvedeno výše, kde Q je polohový vektor, v je rychlost, a je zrychlení, t je čas

a r je poloměr. Protože poslední tři hodnoty jsou úhly, je třeba je převést ze stupňů na radiány.

```
0 references
public void pohybl(Int32 port, string server, double[] Q, string v, string a, string t, string r)
{
    if (client.Connected == false)
    {

        client.Connect(server, port);
    }

    for (int k = 3; k < 6; k++)
    {
        Q[k] = Q[k] * Math.PI / 180;
    }
    network_stream = client.GetStream();
    if (t != "0")
    {
        packet_cmd = utf8.GetBytes("movel(p[" +
        Q[0].ToString("G", CultureInfo.InvariantCulture) + "," +
        Q[1].ToString("G", CultureInfo.InvariantCulture) + "," +
        Q[2].ToString("G", CultureInfo.InvariantCulture) + "," +
        Q[3].ToString("G", CultureInfo.InvariantCulture) + "," +
        Q[4].ToString("G", CultureInfo.InvariantCulture) + "," +
        Q[5].ToString("G", CultureInfo.InvariantCulture) +
        "], t =" + t + ", r=" + r + ")" + "\n");
    }
    else
    {
        packet_cmd = utf8.GetBytes("movel(p[" +
        Q[0].ToString("G", CultureInfo.InvariantCulture) + "," +
        Q[1].ToString("G", CultureInfo.InvariantCulture) + "," +
        Q[2].ToString("G", CultureInfo.InvariantCulture) + "," +
        Q[3].ToString("G", CultureInfo.InvariantCulture) + "," +
        Q[4].ToString("G", CultureInfo.InvariantCulture) + "," +
        Q[5].ToString("G", CultureInfo.InvariantCulture) +
        "], a =" + a + ", v=" + v + ", r=" + r + ")" + "\n");
    }
    network_stream.Write(packet_cmd, 0, packet_cmd.Length);
}
```

Obrázek 5.29: C♯ pohybl

Na obrázku 5.30 je vidět příklad použití. V případě, jak je znázorněno na obrázku, je rychlost nastavena na 1,05 rad/s, zrychlení je nastaveno na 1,4 rad/s^2, čas je nastaven na 0 s a poloměr je nastaven na 0 m. Cílový vektor polohy je `0.2, 0.3, 0.4, 30, 60, 80`. To znamená, že cílová poloha vůči základně je na ose x 200 mm, na ose y 300 mm, na ose z 400 mm, rotace jsou rx 30°, ry 60° a rz 80°.

```
string rychlost = "1.05";
string zrychleni = "1.4";
string t = "0";
string r = "0";
double[] poloha = { 0.2, 0.3, 0.4, 30, 60, 80 };
robot.pohybl(port, server, poloha, rychlost, zrychleni, t, r);
```

Obrázek 5.30: C♯ pohybl použití

Matlab

Na obrázku 5.31 je vidět zdrojový kód funkce pohybl. Nejprve je nutné převést poslední tři čísla pozice ze stupňů na radiány. Příkaz je poté uložen do proměnné „cmd“ a odeslán robotovi.

```
function pohybl(client,q,a,v,t,r)

        q = q.*[1,1,1,pi/180,pi/180,pi/180];
         if t > 0
        cmd =
['movel(p[',num2str(q(1)),',',num2str(q(2)),',',num2str(q(3)),',',num2str(q(4)),',',num2str(q(5)),',',num2str(q(6)),']
,t=',num2str(t),',r=',num2str(r),')'];
        writeline(client,cmd);
        else
        cmd =
['movel(p[',num2str(q(1)),',',num2str(q(2)),',',num2str(q(3)),',',num2str(q(4)),',',num2str(q(5)),',',num2str(q(6)),']
,a=',num2str(a),',v=',num2str(v),',r=',num2str(r),')'];
        writeline(client,cmd);
         end
end
```

Obrázek 5.31: Matlab pohybl

Na obrázku 5.28 je vidět příklad použití v matlabu. Vstupní parametry pro pohybl jsou stejné jako v případě pro C♯ až na rychlost a zrychlení, které jsou 1,1 rad/s a 1,4 rad/s^2.

```
rychlost = 1.1;
zrychleni = 1.4;
t = 0;
r = 0;
poloha = [0.2, 0.3, 0.4, 30, 60, 90];
UR3_lib.pohybl(client, poloha, zrychleni, rychlost, t, r);
```

Obrázek 5.32: Matlab pohybl použití

5.3.3 Speedj

Příkaz speedj je další příkaz k pohybu robota. Po obdržení tohoto příkazu se robot začne pohybovat zadanou rychlostí. To znamená, že pokud robot obdrží příkaz k pohybu základního kloubu určitou rychlostí, pak se základna robota začne otáčet touto rychlostí. Tento příkaz lze použít v například aplikacích, kde je potřeba použití konstantní rychlosti. Vstupy těchto příkazů jsou rychlost, zrychlení, čas. Rychlost je vektor určující rychlost otáčení jednotlivých kloubů. Akcelerace je zrychlení pohybu a čas udává dobu pohybu.

C♯

Obrázek 5.33 ukazuje zdrojový kód funkce pro příkaz speedj. Vstupy do funkce jsou vektor cílových rychlostí, zrychlení a čas dokončení pohybu.

```
1 reference
public void speedj(Int32 port, string server, double[] Qd,  string a, string t)
{
    if (client.Connected == false)
    {

        client.Connect(server, port);
    }
    network_stream = client.GetStream();
    packet_cmd = utf8.GetBytes("speedj([" +
    Qd[0].ToString("G", CultureInfo.InvariantCulture) + "," +
    Qd[1].ToString("G", CultureInfo.InvariantCulture) + "," +
    Qd[2].ToString("G", CultureInfo.InvariantCulture) + "," +
    Qd[3].ToString("G", CultureInfo.InvariantCulture) + "," +
    Qd[4].ToString("G", CultureInfo.InvariantCulture) + "," +
    Qd[5].ToString("G", CultureInfo.InvariantCulture) +
    "], a =" + a +  ", t=" + t  + ")" + "\n");
    network_stream.Write(packet_cmd, 0, packet_cmd.Length);
}
```

Obrázek 5.33: C♯ speedj

Obrázek 5.34 příklad použití je vidět. V případě, jak je to na obrázku, je zrychlení nastaveno na 1,4 rad/s^2 a čas na 0,5 s. Vektor rychlostí je `0.2,0.3,0.5,-0.05,0,0`. To znamená, že základna se bude otáčet rychlostí 0,2 rad/s, rameno 0,3 rad/s, loket 0,5 rad/s, zápěstí 1 při -0,05 rad/s, zápěstí 2 při 0 rad/s a zápěstí 3 při 0 rad/s. Protože je čas nastaven na 0,5 s, pohyb se zastaví po 0,5 s.

```
string zrychleni = "1.4";
string t = "0.5";
double[] rychlosti = { 0.2, 0.3, 0.5, -0.05, 0, 0 };
robot.speedj(port, server, rychlosti, zrychleni, t);
```

Obrázek 5.34: C♯ speedj použití

Matlab

Příklad pro matlab je jako v předchozích příkladech stejný jako pro c♯.

6 Ukázková úloha

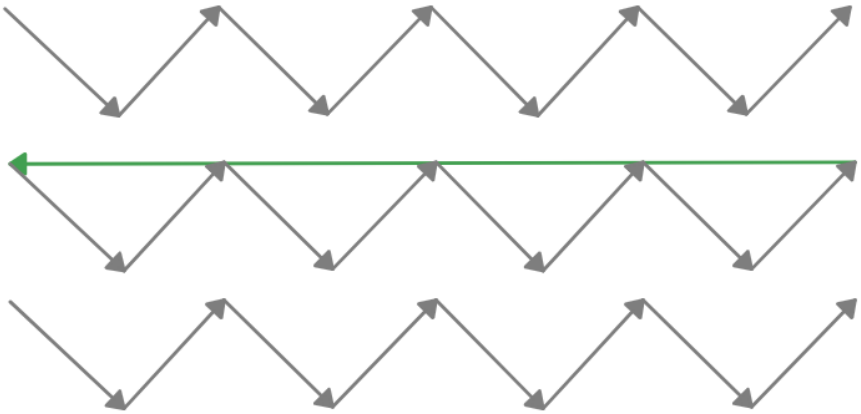
Předchozí kapitola popisovala, jak se připojit k robotu, jak přijímat informace od robota a jak posílat příkazy robotu přes matlab i C♯. Tato kapitola prokáže a ukáže funkčnost programů vytvořených metodami uvedenými v předchozí kapitole.

6.1 Použití programů

Před použitím programů k naprogramování úlohy je nutné zavést programy do řešení. V matlabu se to dělá jednoduše umístěním souboru matlab s programem do stejné složky jako řešení. Poté lze funkce použít zadáním `UR3_lib.FUNKCE_Z_PROGRAMU;`. V C♯ se to provede kliknutím pravým tlačítkem na řešení, najetím na „Přidat“, kliknutím na „existující projekt“ a následným výběrem projektu s programem. K použití funkce je pak potřeba nejprve vytvořit instanci programu v rámci projektu. To lze provést zadáním `UR3_lib.Class1 robot = new UR3_lib.Class1();`. Nyní je možné používat funkce zadáním `robot.FUNKCE_Z_PROGRAMU;`.

6.2 Popis ukázkové úlohy

Cílem ukázkové úlohy je vytvořit program, který bude detekovat a ukládat pohyby robota v reálném čase. Po dokončení tohoto pohybu jej lze opakovat. Jedná se o úkol typu nauč a zopakovat. Obrázek 6.1 ukazuje zjednodušené schéma toho, jak úloha funguje. V horní části obrázku je volně prováděný pohyb. Uprostřed je návrat do výchozí pozice. Ve spodní části je zopakování pohybu.



Obrázek 6.1: Zjednodušené schéma úlohy

6.3 Řešení

Řešení lze rozdělit do tří částí a to deklarace, učení a opakování. V deklarační části je uvedena IP adresa robota, použitý port a zrychlení. V učící části programu se kontroluje skutečné úhlové posunutí kloubů a jejich rychlosti pomocí funkcí `UR3_lib.kloub_natoceni_cil` a `UR3_lib.kloub_rych_skut` a ukládá je. Úhlové posunutí se uloží do proměnné „poloha“ a rychlosti do proměnné „rychlosti“. Maximální počet pozic, které lze uložit, je 65 000. Za předpokladu, že pozice se ukládají každých 8 ms, by to mělo stačit k uložení pohybu dlouhého 8 minut. Pokud se momentální pozice za posledních 200 pozic nezměnila, pak program očekává, že pohyb skončil a přestane kontrolovat a ukládat pozice a rychlosti. V opakovací části program nejprve vyšle příkaz robotu k přesunu do výchozí pozice pomocí funkce `UR3_lib.pohybj`, načež se program pozastaví, aby započítal čas pohybu. Poté robot zopakuje pohyb pomocí uložených hodnot rychlostí a pomocí funkce `UR3_lib.speedj`.

Na obrázku 6.2 lze vidět koncové pozice zopakovaného pohybu v porovnání s koncovou pozicí naučeného pohybu. Na levé straně obrázku je pozice po opakování pohybu a na pravé straně pozice, na kterou robot mířil. Je zřejmé, že hodnoty nejsou stejné. odchylka mezi dvěma polohami je 0,5°. I když se odchylka nezdá tak velká, nepřesnost 0,5° je v robotice nežádoucí. Obrázek 6.4 zobrazuje celý zdrojový kód úlohy.

200.270	199.548
-50.197	-50.722
48.811	49.041
-49.187	-49.007
-21.285	-20.984
439.689	439.768

Obrázek 6.2: Ukázka výsledku

Řešení této úlohy v C♯ je v podstatě stejné jako v matlabu. Největším rozdílem je dynamické přidělování pozic a rychlostí. V matlabu není co řešit, protože si to matlab dělá sám. Způsob dynamického přidělování v C♯ je znázorněno na obrázku 6.3, kde pro dynamickou alokaci pozic a rychlostí byli vytvořeny dva listy po šesti listech.

```
List<List<double>> pozice = new List<List<double>>();
pozice.Add(new List<double> { });
pozice.Add(new List<double> { });
pozice.Add(new List<double> { });
pozice.Add(new List<double> { });
pozice.Add(new List<double> { });
pozice.Add(new List<double> { });

List<List<double>> rychl = new List<List<double>>();
rychl.Add(new List<double> { });
rychl.Add(new List<double> { });
rychl.Add(new List<double> { });
rychl.Add(new List<double> { });
rychl.Add(new List<double> { });
rychl.Add(new List<double> { });
```

Obrázek 6.3: Dynamická alokace v C♯

```
ip = '169.254.62.50';
port = 30003;
client = tcpclient(ip, port);
zrychleni = 1.4;

 while (true)
     dig_vystup = UR3_lib.dig_vystup(client) ;

   while (dig_vystup > 7)
    dig_vystup = UR3_lib.dig_vystup(client) ;
    if (dig_vystup == 9)
       dig_vystup = UR3_lib.dig_vystup(client) ;
       for k =1:65000
           b = skut(:,k-2) -skut(:,k-200);
           if (b == 0 && k > 1500)
             break
           else
             skut(:,k) =UR3_lib.kloub_natoceni_cil(client);
             rychl(:,k) = UR3_lib.kloub_rych_skut(client);
             pause(0.008);
           end
      end
    end


    if (dig_vystup == 10)
          dig_vystup = UR3_lib.dig_vystup(client);
          UR3_lib.pohybj(client, skut(:,1), zrychleni, 1.1,0,0);
          pause(6);
          a = 1;
         for k =1:5:length(rychl)
           UR3_lib.speedj(client, rychl(:,k), zrychleni,0.09);
           pause(0.06);
         end
     end
  end
end
```

Obrázek 6.4: Zdrojový kód pro řešení úlohy v matlabu

7 Závěr

Tato bakalářská práce ukazuje možné řešení vytvoření programu pro dálkové ovládání robota UR3. Komunikace s robotem byla nastavena pomocí protokolu TCP/IP. Zvoleným komunikačním rozhraním je real time port 30003 a zvolenými programovacími rozhraními jsou Matlab a C♯. Výsledkem této práce jsou dva programy, jeden v matlabu a jeden v C♯, které obsahují 35 funkcí pro zjištění stavu robota a 8 příkazových funkcí. Funkčnost těchto programů byla ukázána na příkladu úlohy na reálném robotu.

Při řešení této bakalářské práce se vyskytlo několik problémů. První z nich byla rozdílná verze Real time robota v URsim a skutečného robota. V URsim byla verze 5.9, ale skutečný robot měl verzi 5.10, což vedlo k nesprávné délce přijatých zpráv při dotazu na stav robota. To způsobilo, že informace o stavu robota byly nepřesné. Další problém byl při řešení příkladové úlohy. Pohyb robota při opakování pohybu byl spíše trhaný než plynulý. To bylo vyřešeno pomocí příkazu speedj. Ani toto řešení však není ideální, protože přináší problém s přesností.

Možným vylepšením této práce by mohlo být zahrnutí všech příkazů v rámci URscriptu. Vytvoření algoritmu, který by v ukázkové úloze našel důležité body v rámci pohybu, jako jsou například body, ve kterých se mění směr pohybu a poté pohybovat robota podle těchto bodů.

Jak bylo zmíněno, programy byly vytvořeny v matlabu a c♯. Při porovnání těchto dvou by optimálnější volbou pro ovládání robota byl matlab. Matlab byl vytvořen s ohledem na vektory a pole, což usnadňuje ovládání robota. Nicméně pokud jde o komerční použití, C♯ je jasným vítězem díky svému bezplatnému použití. Pro shrnutí je srovnávací matlab lepší volbou, pokud jde o nekomerční oblasti, jako je akademická oblast, a C♯ je lepší volbou, pokud jde o komerční oblasti.

Použitá literatura

[1] ING. JOSEF ČERNOHORSKÝ, Ph.D. doc. *Základy robotiky* [online]. Přednášky. TU v Liberci, 2020 [cit. 04. 05. 2022]. Dostupné z: https://elearning.tul.cz/course/view.php?id=7851.

[2] *Ur3* [online]. Universal Robots [cit. 03. 05. 2022]. Dostupné z: https://www.universal-robots.com/3d/images/slider/ur3/small_images/rendersd_00008.jpg.

[3] *User manual: UR3/CB3* [online]. Universal Robots, 2021 [cit. 30. 04. 2022]. Dostupné z: https://s3-eu-west-1.amazonaws.com/ur-support-site/105316/99341_UR3_User_Manual_en_E67ON_Global.pdf.

[4] *Ur kontrolér* [online] [cit. 03. 05. 2022]. Dostupné z: https://automatizace.hw.cz/files/images/ur_controller01.jpg.

[5] *Ur kontrolér kontakty* [online]. Universal Robots [cit. 03. 05. 2022]. Dostupné z: https://www.universal-robots.com/media/1812215/articles_01.png.

[6] *The URScript Programming Language* [online]. Universal Robots, 2018 [cit. 30. 04. 2022]. Dostupné z: https://s3-eu-west-1.amazonaws.com/ur-support-site/32554/scriptManual-3.5.4.pdf.

[7] *Realtime Client Interface* [online]. Universal Robots, 2021 [cit. 30. 04. 2022]. Dostupné z: https://s3-eu-west-1.amazonaws.com/ur-support-site/16496/ClientInterfaces_Realtime.pdf.

[8] *Guide:Setting up the virtual environment to simulate UR robots* [online]. Universal Robots, 2020 [cit. 07. 05. 2022]. Dostupné z: https://academy.universal-robots.com/media/jiehhszc/ursim_vmoracle_installation_guidev03_en.pdf.

[9] *Virtualbox*. Redwood Shores, CA, 1995. Dostupné také z: https://www.virtualbox.org/.

[10] *Instrument Control Toolbox: User's Guide* [online]. The MathWorks, Inc., 2022 [cit. 29. 04. 2022]. Dostupné z: https://www.mathworks.com/help/pdf_doc/instrument/instrument_ug.pdf.

[11] MILLER, Mark R.; MILLER, Rex. *Robots and robotics: principles, systems, and industrial applications*. New York: McGraw-Hill Education, [2017]. ISBN 978-1-259-85978-6.

Přílohy

A-CD

- Text bakalářské práce ve formátu PDF
- Program s funkcemi pro ovládání robota v Matlabu
- Program s funkcemi pro ovládání robota v C♯
- Program ukázkové úlohy v Matlabu
- Program ukázkové úlohy v C♯