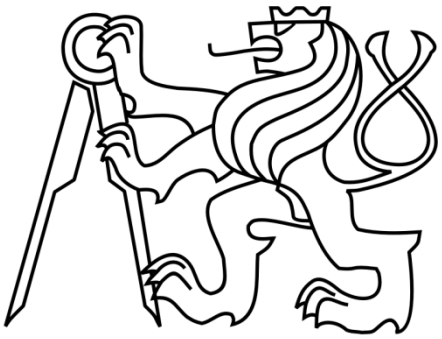
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

FAKULTA STROJNÍ

ÚSTAV MECHANIKY, BIOMECHANIKY A MECHATRONIKY

Odbor mechaniky a mechatroniky



# Bakalářská práce

## Zprovoznění a kalibrace robota uArm Swift Pro

**Praha, 2019** **Martin Jílek**

# ZADÁNÍ BAKALÁŘSKÉ PRÁCE

## I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Jílek** Jméno: **Martin** Osobní číslo: **466681**

Fakulta/ústav: **Fakulta strojní**

Zadávající katedra/ústav: **Ústav mechaniky, biomechaniky a mechatroniky**

Studijní program: **Teoretický základ strojního inženýrství**

Studijní obor: **bez oboru**

## II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

**Zprovoznění a kalibrace robota uArm Swift Pro**

Název bakalářské práce anglicky:

**Commissioning and calibration of the uArm Swift Pro robot**

Pokyny pro vypracování:

1) Seznamte se s konstrukcí robotického ramene uArm Swift Pro.
2) Zprovozněte ovládání robota v prostředí ROS.
3) Připravte kinematický model robota.
4) Proveďte kinematickou kalibraci robota.

Seznam doporučené literatury:

[1] Stejskal, V., Valášek, M.: Kinematics and Dynamics of Machinery, Marcel Dekker, Inc., New York 1996.
[2] Šika, Z., Hamrle, V., Valášek, M., Beneš, P.: Calibrability as Additional Design Criterion of Parallel Kinematic Machines, Mech. Mach. Theory, 50, 2012, s. 48–63.

Jméno a pracoviště vedoucí(ho) bakalářské práce:

**Ing. Petr Beneš, Ph.D., odbor mechaniky a mechatroniky FS**

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **29.04.2019** Termín odevzdání bakalářské práce: **16.08.2019**

Platnost zadání bakalářské práce: _____

| Ing. Petr Beneš, Ph.D. | prof. Ing. Milan Růžička, CSc. | prof. Ing. Michael Valášek, DrSc. |
|---|---|---|
| podpis vedoucí(ho) práce | podpis vedoucí(ho) ústavu/katedry | podpis děkana(ky) |

## III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

Datum převzetí zadání Podpis studenta

# Anotační list

| | |
|---|---|
| **Jméno autora:** | Martin Jílek |
| **Název bakalářské práce:** | Zprovoznění a kalibrace robota uArm Swift Pro |
| **Anglický název:** | Commisioning and calibration of the uArm Swift Pro robot |
| **Akademický rok:** | 2018/2019 |
| **Obor studia:** | Teoretický základ strojního inženýrství |
| **Ústav/odbor:** | Ústav mechaniky, biomechaniky a mechatroniky<br>Odbor Mechaniky a mechatroniky |
| **Vedoucí bakalářské práce:** | Ing. Petr Beneš, Ph.D. |
| **Bibliografické údaje:** | Počet stran: 40<br>Počet obrázků: 23<br>Počet příloh: 11 |
| **Klíčová slova:** | uArm Swift Pro, kinematická kalibrace, ROS |
| **Keywords:** | uArm Swift Pro, Kinematic calibration, ROS |

**Anotace:**

Obsahem této bakalářské práce je seznámení se s konstrukcí robotického ramene uArm Swift Pro a jeho následné zprovoznění v prostředí ROS (Robot Operating System). Dalším bodem je připravení kinematického modelu a kinematická kalibrace.

**Abstract:**

The aim of the thesis is introduction to the construction of the uArm Swift Pro robot arm followed by its commisioning in ROS environment (Robot Operating System). Next step is preparation of the kinematic model and kinematic calibration.

## Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně a použil jsem pouze podklady uvedené v přiloženém seznamu.

V Praze, dne …… ……

Podpis

## Poděkování

Rád bych poděkoval svému vedoucímu bakalářské práce, Ing. Petru Benešovi, Ph.D, za pomoc při zpracování této bakalářské práce. Dále bych chtěl poděkovat svým rodičům, kteří mě po celou dobu studia plně podporovali.

# Obsah

## Seznam obrázků



## Seznam tabulek

# 1. Úvod

V dnešním světě plném automatizace je kladen důraz na rychlost, preciznost a nízkou nákladovost výroby. Robotické manipulátory již vytlačily manuální pracovníky ve spoustě odvětví, nejvíc je to však znatelné v automotive průmyslu. Tyto manipulátory dokáží provést daleko náročnější operace než běžný člověk s mnohem větší přesností, rychlostí a opakovatelností. I přes počáteční vysoké náklady se investice do těchto strojů v dlouhodobém měřítku vyplatí a pro udržení konkurenceschopnosti a požadované přesnosti výroby některých dílů je dokonce nezbytná.

Všechny tyto manipulátory, obráběcí centra a robotická ramena jsou řízeny sofistikovanými algoritmy, které kombinují znalosti z klasické mechaniky, elektroniky a počítačového řízení. Avšak i ten nejdokonalejší algoritmus musí počítat s určitými výrobními odchylkami, které jsou však pro každý vyrobený kus, každou součást celého manipulátoru rozdílné. Jak tedy zjistit přesné rozměry konkrétního stroje, aby řídící systém dokázal i přes nepřesnosti korektně a spolehlivě provádět svůj úkol? V tento moment přichází na scénu kalibrování každého manipulátoru.

Kalibrace je proces, který se snaží co nejpřesněji určit rozměry daného stroje, aby se skutečná poloha akčního prvku (tzv. end effector) co nejméně lišila od požadované hodnoty. Při kalibrování jsou pro nás počáteční data souřadnice v určitých námi zvolených polohách. Obecně platí, čím více poloh změříme, tím více rovnic získáme a řešení bude tedy přesnější. Musí se však dávat pozor na konvergenci řešení, aby výpočet byl dostatečně efektivní. Samozřejmě, že i samotný proces měření pomocí externího měřidla s sebou přináší další nepřesnosti, tím se však tato bakalářská práce nezabývá.

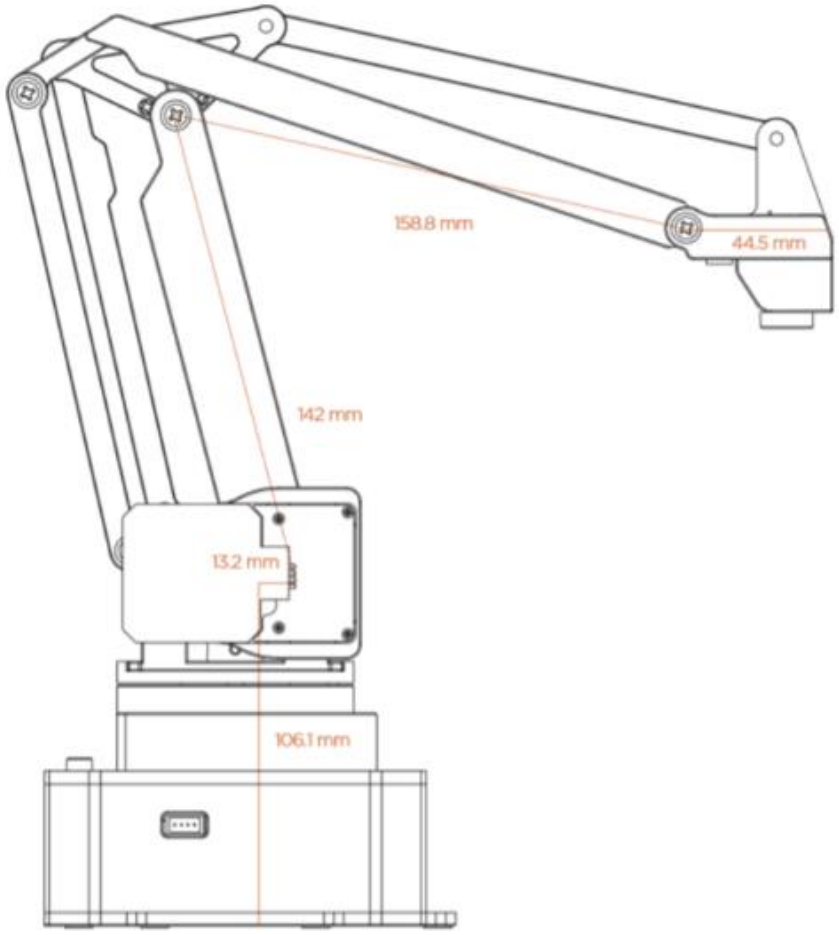
Robotické rameno uArm Swift Pro je ideální pro seznámení se základními principy robotiky. Podporuje řadu programovacích jazyků a lze s ním provádět základní úkony. Je malé, lze s ním tedy snadno manipulovat. Navíc je poměrně levné, může si ho tedy pořídit téměř každý.

# 2. Cíle

Mezi cíle patří seznámení se s prostředím ROS, jeho strukturou a potenciálem využití jeho komponent. Těchto znalostí bylo využito při zprovoznění robota v prostředí ROS, což byl i hlavní cíl práce. I přesto, že toto prostředí je poměrně nové, jádro a základní principy funkce se již ustálily a přibývají pouze nadstavbové komponenty, které člověk může využít pouze v případě jejich potřeby a do základních funkcí na nejnižší úrovni programu nezasahují. Prostředí ROS má obrovský potenciál, jak v akademickém prostředí, tak průmyslu, protože si každý může funkci přizpůsobit svým potřebám a možnosti využití jsou prakticky neomezené. Dílčím cílem bylo sestavit kinematický model, provést kinematickou kalibraci a pochopit princip jejího fungování.

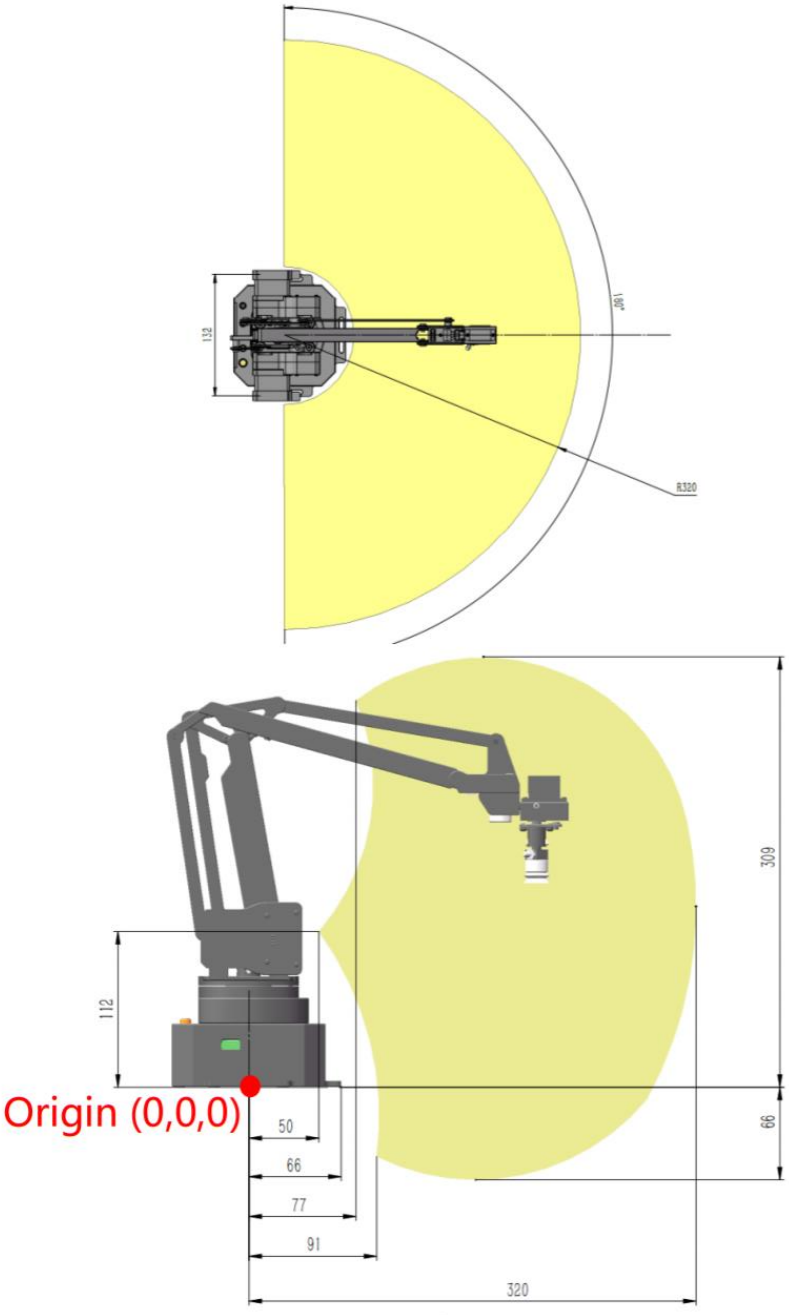
# 3. Seznámení s konstrukcí robotického ramena uArm Swift Pro

Tento robot není určen pro průmyslové použití, je to spíše výukový model, na kterém si mohou začátečníci vyzkoušet základní principy robotiky. K tomu je dodáván software uArm Studio, který je intuitivním uživatelským prostředím, kde programování probíhá pomocí blokových schémat. Dále podporuje i Arduino, Python, GRABCAD a právě ROS.




Obr. 1 – Schéma robota [2]

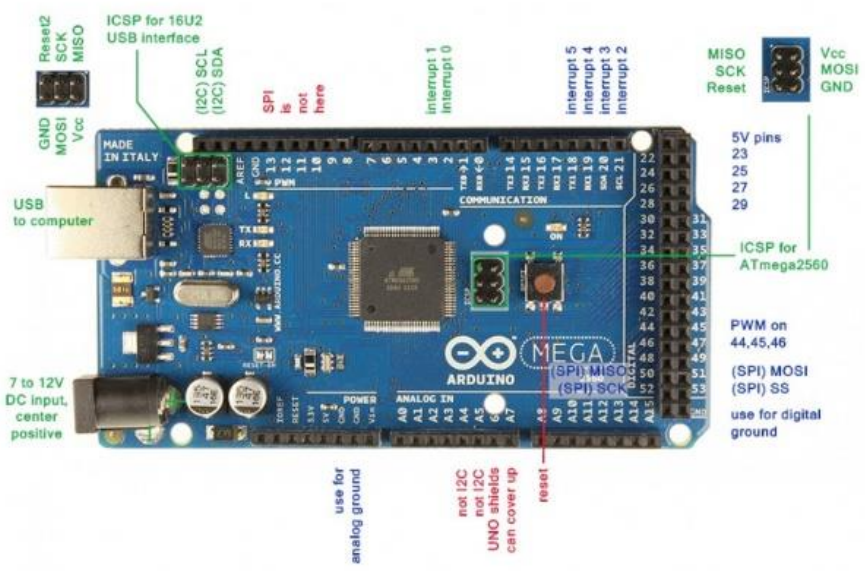
Robot má celkem 4 stupně volnosti, přičemž jeden je vyhrazen pro koncový efektor. Tím může být klasický gripper, přísavka, laser nebo 3D tisková hlava. Výrobce udává přesnost pohybu 0,2 mm a schopnost zvednout objekt až o váze 500 g. Na obrázku 1 je schéma robota se základními funkčními kótami.




Obr. 2 - Pracovní oblast [21]

Na obrázku 2 je vidět pracovní oblast robotického ramene s vyznačením počátku souřadného systému. Z kinematiky vyplývá, že koncový bod s akčním členem je v každé poloze vodorovný, toho je docíleno sekundárním přepákováním a přidáním paralelogramu.

Základem je deska Arduino Mega 2560, která obsahuje mikroprocesor ATmega 2560 s taktovací frekcencí 16 MHz. Flash paměť sloužící k ukládání kódu je 256 kB a dočasná paměť RAM je 8 kB. Deska obsahuje 52 digitálních I/O pinů, 16 analogových vstupů a rozhraní USB. Pracuje s napětím 5 V a vstupní napětí je 7-12 V. Jde v podstatě o zvětšeninu populární desky Arduino UNO. Liší se zejména větším počtem pinů a větším výkonem. Doménou prostředí Arduino je snadné připojení k počítači a jednoduché programování. Jazyk je syntaxí podobný jazyku C, používá však svoje procedury a celý program běží ve smyčce (loop). [1]




Obr. 3 – Deska Arduino Mega 2560 [20]

Samotný uArm Swift Pro váží 2,2 kg a k počítači je připojen přes MicroUSB. Podporuje však i Bluetooth a je tedy možné bezdrátové řízení. Rozdíl mezi základní verzí a verzí uArm Swift Pro je zejména v motorech. Swift Pro obsahuje vlastní převodovku a krokové motorky, zatím Swift má pouze servomotory. Model Swift Pro navíc pro zjištění konkrétní pozice používá 12-bitový enkodér. Enkodér je snímač polohy motoru a většinou se používá jako zpětná vazba, např. u CNC strojů. Dělí se na inkrementální (při jejich použití musí stroj při každém zapnutí znovu najíždět na kalibrační čidlo, aby souhlasilo odměřování polohy) a absolutní, ty si i po vypnutí stroje pamatují svou polohu. Model Swift ke zjištění polohy používá potenciometr. Kroutící moment motorů je 12 $kg \cdot cm$. [2]

# 4. Zprovoznění v prostředí ROS

## 4.1. Seznámení s ROSem

ROS byl vytvořen v roce 2007 na Stanfordské univerzitě a od roku 2013 je spravován OSRF (Open Source Robotics Foundation). V dnešní době je používán na řadě univerzit i ve firmách k řízení robotů různých druhů a účelů.

ROS je zkratka pro Robot Operating System. Sestává se ze 4 klíčových elementů. Prvním je tzv. plumbing, což vlastně znamená, že naráz běží více programů a procedur, které mohou být psány v různých jazycích (C++, Python, MATLAB, Java) a ROS zprostředkovává komunikaci mezi nimi. Druhým elementem je pestrá škála pracovních balíčků (tools), jako např. vizualizéry a grafický interface. Třetím elementem jsou tzv. capabilities. Ty zajišťují ovládání, plánování, manipulaci atd. Nespornou výhodou je, že celý ekosystém ROSu je open-source, mohu tedy brát balíčky již vytvořené jinými lidmi a pouze je přizpůsobovat svému konkrétnímu použití. Tím se dostávám ke čtvrtému elementu, čímž je rozsáhlá komunita, která spravuje obsáhlou wiki, dokumentaci a tutoriály pro přiblížení základních komponent ROSu začátečníkům. [3]

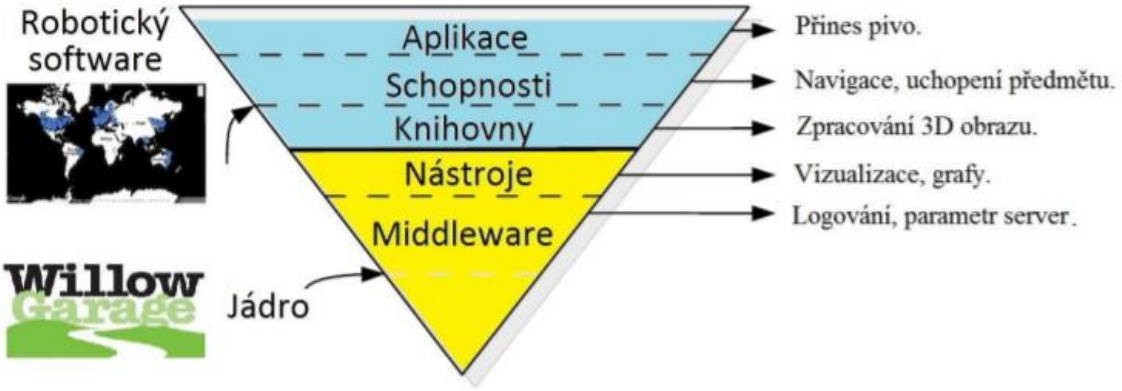
ROS je založen na technologii peer to peer. To je typ sítě, kde spolu komunikují jednotliví klienti. Opakem je typ klient-server, kde každý klient komunikuje se serverem a přes něj s ostatními klienty. V případě ROSu spolu klienti komunikují přes ROS messages nebo ROS services. To navíc umožnuje, že jednotlivé programy mohou běžet na více počítačích a komunikovat spolu přes síť. [4]

Známkou podtrhující rozšiřující se oblíbenost ROSu je také mezinárodní workshop ROSCon, který se koná každoročně od roku 2012. V roce 2019 se bude konat 1. listopadu v Macau.

Jedním z hlavních cílů ROSu je snaha vytvořit flexibilní a zároveň robustní software pro řízení rozličných robotů a problémů s tímto spojených. Vysoká modularita umožňuje uživatelskou volbu, zda si určitou část kódu vytvoří sám, či využije nějakou z již hotových komponent a přizpůsobí ji svým potřebám. [5]

Jelikož se ROS neustále vyvíjí, jsou velké změny soustředěny do tzv. distribucí. Jde vlastně o aktualizace verze balíčků. V době psaní této práce (první polovina roku 2019) je nejnovější distribucí ROS Melodic Morenia, ve kterém probíhala i veškerá práce.

Vzhledem k bohaté struktuře bývá ROS označován také jako ekosystém, dělící se na jádro a robotický software. Jádro je vyvíjeno a spravováno společností Willow Garage a balíčky jsou vytvářeny rozsáhlou komunitou.




Obr. 4 - Ekosystém ROSu [5]

Jádro poskytuje zejména správu balíčků, což umožňuje měnit velikost systému vzhledem k potřebám uživatele. Distribuce balíčků probíhá přes repositories, které bývají ve většině internetové servery, kam jsou balíčky uloženy. Další částí je tzv. Middleware, fungující na bázi „softwarového lepidla“, umožňující propojení a kompatibilitu různého softwaru napříč platformami. Poskytuje služby jako zasílání messages (zpráv), Parameter server, knihovny atp. [5]

K rozhýbání robota Swift Pro byly použity balíčky pro ROS poskytované přímo výrobcem uArm [6]. Jelikož ROS není samostatný operační systém, veškerá práce probíhala v prostředí Linux Ubuntu, existuje však i emulátor do prostředí Microsoft Windows, ten však ze své osobní zkušenosti nedoporučuji, neboť má problémy při správě hardwaru. Např. sériové porty jsou pod operačním systémem Linux značeny jinak (ttyACMO) než ve Windowsovém prostředí (COM1, COM2 atd.), a protože je ROS určen pro Linux, nedokáže si poradit s odlišným značením. Následující kapitoly věnuji instalaci, popsání základních funkcí a komponent nutných k řízení robota v ROSu.

### 4.2. Instalace a zprovoznění robota

Nejprve je potřeba nastavit počítač tak, aby akceptoval software z package.ros.org a to následujícím příkazem v terminálu:

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'
```

Následně se musí nastavit klíč potřebný k připojení k serveru.

```
sudo apt-key adv --keyserver 'hkp://keyserver.ubuntu.com:80' --recv-key C1CF6E31E6BADE8868B172B4F42ED6FBAB17C654
```

Nyní již samotná instalace. Plná instalace obsahuje ROS, rqt (což je framework pro grafické uživatelské prostředí) a nezbytné knihovny.

```
sudo apt install ros-melodic-desktop-full
```

V případě, že uživateli tyto balíčky nestačí, následujícím příkazem lze vyvolat seznam všech dostupných balíčků.

```
apt search ros-melodic
```

Před prvním použitím ROSu je nutné inicializovat *rosdep.* Ten je potřebný pro běh určitých komponent ROSu.

```
sudo rosdep init
rosdep update
```

Je vhodné, aby se proměnné používané ROSem automaticky přidávaly do uživatelského *bash* souboru při každém spuštění terminálu.

```
echo "source /opt/ros/melodic/setup.bash" >> ~/.bashrc
source ~/.bashrc
```

Do této chvíle byly nainstalovány hlavní balíčky, které jsou nutné ke správnému chodu. K vytváření a spravování vlastních balíčků existuje řada nástrojů, které jsou distribuované zvlášť. Ke stažení některých balíčků umožňujících tyto funkce byl použit příkaz:

```
sudo apt install python-rosinstall python-rosinstall-generator python-wstool build-essential
```

### 4.3. Stažení balíčků pro ovládání robota

Nejprve je potřeba se přesunout do složky, kde je aktuální workspace příkazem v terminálu:

```
$ cd ~/catkin_ws/src
```

Následně stáhneme balíčky do této složky příkazem:

```
$ git clone https://github.com/uArm-Developer/RosForSwiftAndSwiftPro.git
```

Poté balíčky zkompilujeme, to už je příkaz ROSu. Tím vytvoříme požadovanou strukturu.

```
$ catkin_make
```

Posledním krokem je úprava *bash* souboru. Tím se ROS nastaví pro práci na tomto projektu.

```
echo "source /opt/ros/kinetic/setup.bash" >> ~/.bashrc
echo "source ~/catkin_ws/devel/setup.bash" >> ~/.bashrc
source ~/.bashrc
```

Tím je vše připraveno a je možno přistoupit k simulacím tohoto robota v prostředí ROS.

### 4.4. Ovládání robota v prostředí ROS

K ovládání robota je potřeba propojit počítač s nainstalovaným ROSem k robotu přes datový USB kabel. Následně v terminálu povolíme přístup přes port ttyACM0, který v prostředí Ubuntu Linux slouží jako port pro sériovou komunikaci.

```
$ sudo chmod 666 /dev/ttyACM0
```

K simulaci aktuálního stavu robota v prostředí RViz slouží následující příkaz. ROS čte data z pohonů a Rviz vykresluje stav na základě těchto údajů.

```
roslaunch swiftpro pro_display.launch
```

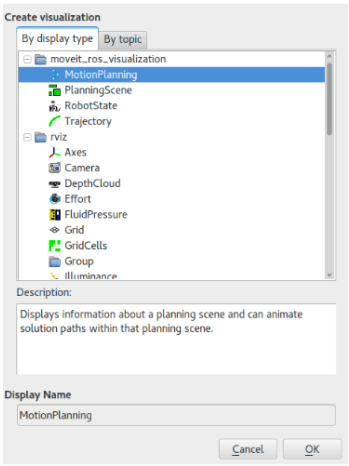
V ovládacím módu tečou data opačnou stranou, tedy z počítače k robotu. Toho se docílí příkazem:

```
roslaunch swiftpro swift_control.launch
```

Následně je potřeba spustit komponentu MoveIt, ve které se provádí plánování trajektorie.

```
roslaunch swift_moveit_config demo.launch
```

V prostředí Rviz je potřeba přidat komponentu „Motion Planning“ v levém okně.

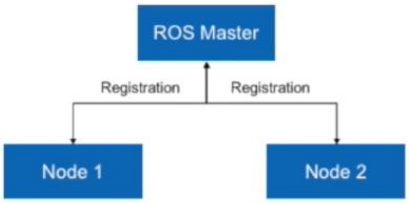



Poté lze tažením os určit počáteční a koncovou polohu robota. Skutečnou barvou je vyobrazena referenční poloha robota. Zeleně počáteční poloha pohybu a oranžově pak koncová poloha. Po dokončení plánování a nastavení počáteční a koncové polohy do zvolených pozic MoveIt podle kinematického popisu robota vypočítá trajektorii potřebnou k uskutečnění tohoto pohybu. Přitom dbá na možné kolize a snaží se jim vyhnout. Zároveň kontroluje, zda se nějaká součást nedostala mimo pracovní prostor, kam nemůže fyzicky dosáhnout.

## 4.5. Komponenty a struktura ROSu

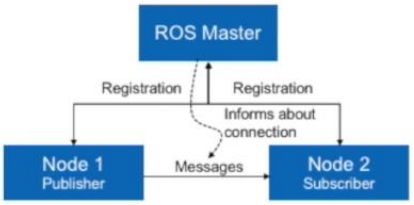
### 4.5.1. ROS Master a ROS Nodes

ROS Master spravuje komunikaci mezi uzly. Princip funkce je naznačen na Obr. 5. Master se spustí příkazem *roscore*. Jeho hlavním účelem je, aby se jednotlivé uzly (nodes) dokázaly najít ve struktuře ROSu.




Obr. 5 – ROS Master [24]

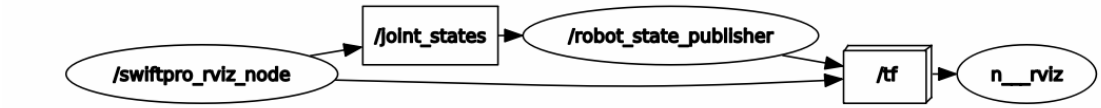
ROS Nodes (uzly) jsou jednoúčelové programy, které jsou samostatně kompilovány, vykonávány a spravovány. Jsou sdružovány do balíčků (packages). Každý uzel se musí registrovat k Masteru, jinak nebude propojen se sítí a nebude vykonávat svou funkci. Jednotlivé uzly spolu komunikují přes ROS Topics, což je proud zpráv mezi uzly. Uzly mohou tzv. publikovat (publish), nebo odebírat (subscribe) jednotlivé topicy. Rozdíl je ve způsobu komunikace. Publisher odesílá data subscriberu. Tento způsob komunikce je naznačen na Obr. 6. [4]




Obr. 6 – Komunikace mezi uzly [24]

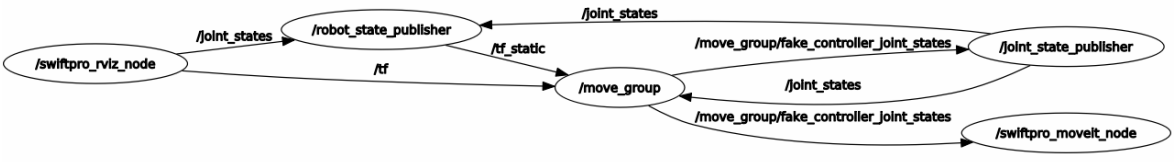
Většinou existuje 1 publisher a N subscriberů. ROS Message je datová struktura definující typ topicu. Obsahuje proměnné typu integer, float, boolean, string nebo array. Více v kapitole Ros messages (zprávy) .

Na Obr. 7 je vidět graf komunikace robota Swift Pro při získávání dat z jednotlivých linků. Tento graf je generován pomocí pluginu rqt_graph. V oválných bublinách jsou vypsány uzly (nodes) a v obdélnících topicy. Směr šipek naznačuje, který člen v hierarchii je subscriber a který publisher (v tomto případě je uzel /swiftpro_rviz_node publisher a /robot_state_publisher mu subscribuje přes topic /joint_states). Balíček (package) /tf zaznamenává více souřadných systémů (každý link má svůj souřadný systém) a spravuje vztahy mezi nimi. Dále umožnuje uživateli přepočítávat data mezi jednotlivými souřadnými systémy.




Obr. 7 – Komunikace mezi uzly (nodes) a topicy

Na Obr. 8 je znázorněna komunikace mezi uzly při ovládání robota přes Motion Planning komponentu balíčku MoveIt. Jeho hlavním úkolem je poskytnout potřebné trajektorie pro robotické rameno za účelem vyslání koncového efektoru na určité místo. Samotné plánování pohybu robota je velice náročné, protože je potřeba vypočítat sadu hodnot, které každý kloub musí vykonat v souladu s ostatními klouby.




Obr. 8 – Schéma komunikace při motion planningu

### 4.5.2. ROS Launch

ROS Launch je nástroj pro spouštění uzlů a nastavování parametrů. Jsou psány v jazyce XML [7]. Na Obr. 9 je vidět kód mého. Slouží ke spuštění uzlů pro řízení robota Swift Pro.

```
<launch>
        <param name="robot_description" command="cat $(find swiftpro)/urdf/pro_model.xacro" />
        <param name="use_gui" value= "False" />

        <node name="swiftpro_write_node" pkg="swiftpro" type="swiftpro_write_node"/>
        <node name="swiftpro_moveit_node" pkg="swiftpro" type="swiftpro_moveit_node"/>
        <node name="swiftpro_rviz_node" pkg="swiftpro" type="swiftpro_rviz_node"/>

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

Obr. 9 - ROS Launch [6]

Uzel swiftpro_moveit_node obsahuje dopřednou kinematiku pro dopočítání koncového bodu ramene a uzel swiftpro_rviz_node obsahuje inverzní kinematiku pro vykreslení aktuálního stavu robota v prostředí RViz. Uzel swiftpro_write_node obdrží souřadnice v kartézských souřadnicích, převede je do G kódu a odešle přes sériový port.

### 4.5.3. Ros messages (zprávy)

Podle Obr. 6 je komunikace mezi uzly realizována přes messages (zprávy). Každá zpráva je definována svým jménem a obsahuje určitou datovou strukturu. Navíc je možné vytvořit z obsažených datových struktur pole jak statické, tak i dynamické. Zpráva má příponu *msg* a v balíčku je umístěna v adresáři msg. Zpráva SwiftproState.msg obsahuje všechna data o mém konkrétním robotu v konkrétní poloze. [8]

```
float64 motor_angle1
float64 motor_angle2
float64 motor_angle3
float64 motor_angle4
float64 x
float64 y
float64 z
uint8   pump
uint8   swiftpro_status
uint8   gripper
```

Obr. 10 - SwiftProState.msg [6]

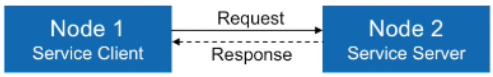
V Tab. 1 je popis funkce rosmsg, která slouží jako diagnostika zpráv.

| funkce | 1. argument | 2. argument | popis |
|---|---|---|---|
| *rosmsg* | show | Název_zprávy | Zobrazí popis zprávy |
| | list | - | Zobrazí všechny zpávy |
| | package | Název balíčku | Zobrazí zprávy v balíčku |
| | packages | Název_zprávy | Zobrazí balíčky obsahující zprávu |

Tab. 1 – Použití funkce *rosmsg* [9]

### 4.5.4. ROS Services (služba)

Na rozdíl od vztahu subscriber/publisher se ROS Services zakládají na request (požadavku) a response (odpovědi) mezi jednotlivými uzly, kde Service server odpovídá na požadavky klientovi. Services mají podobnou strukturu jako messages, avšak mají příponu *srv*. Oproti tomu existuje ROS Actions, kde server navíc dostává zpětnou vazbu na průběh úkolu a ten jde zrušit ještě před jeho ukončením. [10]




Obr. 11 – ROS Services [25]

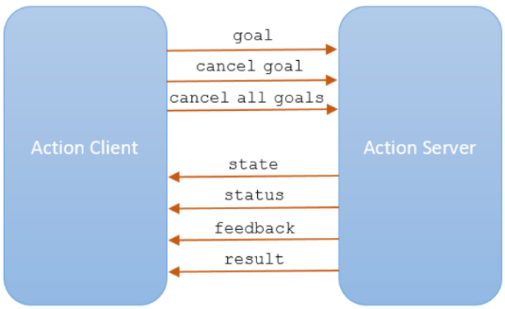
Používání ROS services a argumenty sloužící k jejich vyvolání jsou shrnuty v Tab. 2.

| funkce | 1. argument | 2. argument | 3.argument | popis |
|---|---|---|---|---|
| *rosservice* | args | Název_zprávy | - | Zobrazí argumenty služby |
| | call | Název_zprávy | Argumenty služby | Zobrazí služby s argumenty |
| | find | Název balíčku/zprávy | - | Najde službu |
| | list | - | - | Zobrazí seznam služeb |
| | info | Název_zprávy | - | Zobrazí informace o službě |
| | node | Název_zprávy | - | Zobrazí uzel |
| | type | Název_zprávy | - | Zobrazí typ služby |
| | uri | Název_zprávy | - | Zobrazí URI adresu |

Tab. 2 – Používání funkce *rosservice* [10]

### 4.5.5. ROS Actions

ROS Actions komunikují jako klient-server přes specifický protokol. Přizpůsobují ROS Topics k vyslání požadavku od klienta na server. Tyto požadavky lze v průběhu zrušit. Po přijmutí požadavku server zpracuje informaci a pošle zpětnou vazbu klientovi. Tato zpětná vazba obsahuje stav serveru a aktuálního požadavku a výslednou zprávu, pokud je požadavek splněn. [11]




Obr. 12 - ROS Actions [11]

### 4.5.6. ROS Packages

Software ROSu je organizován do balíčků (packages). Každý balíček obsahuje zdrojový kód, launch soubory, konfigurační soubory, messages a dokumentaci. Balíček, který se vztahuje k jinému balíčku, nebo na něm nějak závisí, je propojen přes tzv. dependencies. Ty jsou realizovány v Manifests, což je vlastně soubor ve formátu XML obsahující popis balíčku. Příklady takovýchto balíčků jsou např. výše zmíněný MoveIt, nebo TF. [12]

### 4.5.7. ROS Parameter server

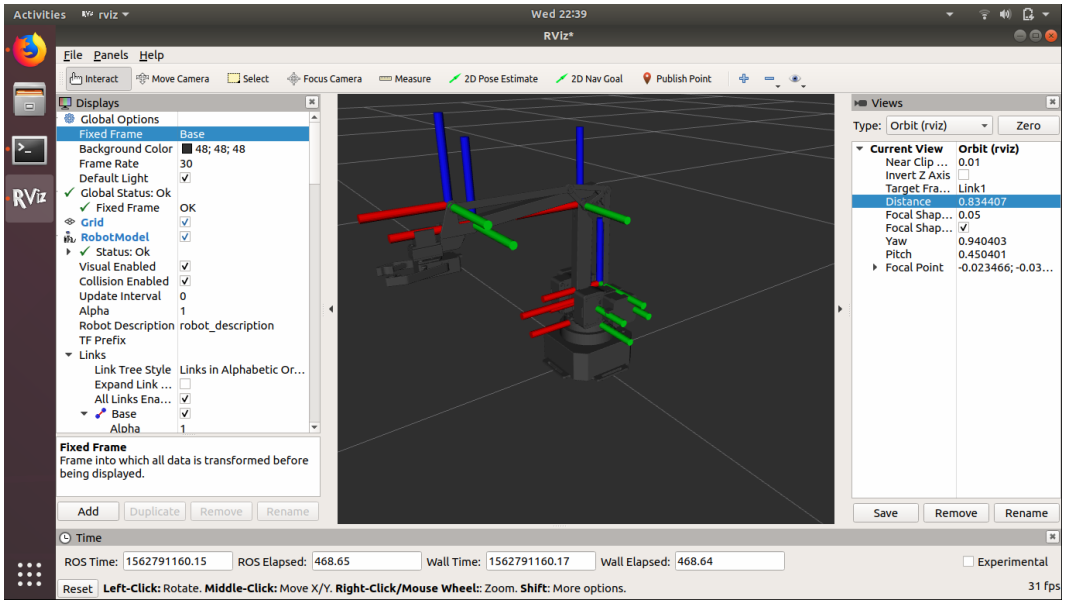
Parameter server je sdílený slovník přístupný přes síťovou API. Uzly používají tento server k ukládání a získávání parametrů za běhu. Ty mohou být různých datových typů jako např. integer, boolean, string, double atd. Jelikož není tvořen pro vysoký výkon a náročné operace, je nejlépe využitelný pro statická data jako např. parametry konfigurací atp. Parametr server je implementován v ROS Masteru. Tab. 3 obsahuje možnosti funkce *rosparam.* [13]

| funkce | 1. argument | 2. argument | 3. argument | popis |
|---|---|---|---|---|
| *rosparam* | set | Název_parametru | Hodnota_parametru | Nahraje parametr s konkrétní hodnotou |
| | get | Název_parametru | - | Získá parametr |
| | load | Název_souboru | - | Nahraje parametr |
| | dump | Název_souboru | - | Smaže soubor |
| | delete | Název_parametru | - | Smaže parametr |
| | list | - | - | Vypíše parametry |

Tab. 3 - Používání funkce *rosparam* [13]

### 4.5.8. RViz

RViz slouží jako 3D vizualizace pro ROS a je obsažen v již základní instalaci. Funguje tak, že subscribuje topicy a vizualizuje obsah messages mezi nimi. Nabízí různé pohledy kamery (ortografická, horní, boční) a je rozšiřitelný o řadu pluginů. Podporuje zobrazení jednotlivých linků robota a vypisuje informace o pozici jednotlivých linků ve svých lokálních souřadných systémech. Robot je vykreslován pomocí pluginu Robot model. [14]




Obr. 13 – Rviz vizualizace

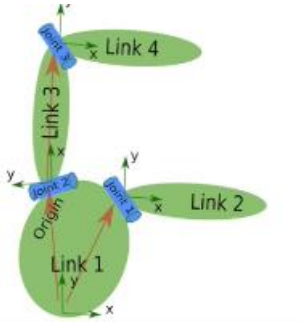
### 4.5.9. Gazebo

Gazebo je alternativa k RVizu, slouží tedy jako 3D grafický simulátor, navíc obsahuje svůj vlastní fyzikální engine. Umožňuje virtuální otestování aplikace před

implementací ve skutečném světě. [5] Dále umožňuje přesně a efektivně simulovat chování skupin robotů v různých prostředích. [15]

### 4.5.10. Robot Model

Robot je složen z tzv. linků a jointů (kloubů). Klouby spojují jednotlivé linky k sobě. Tím je vlastně popsána kinematická struktura robota.




Obr. 14 – Možný kinematický popis robota se 4 linky [22]

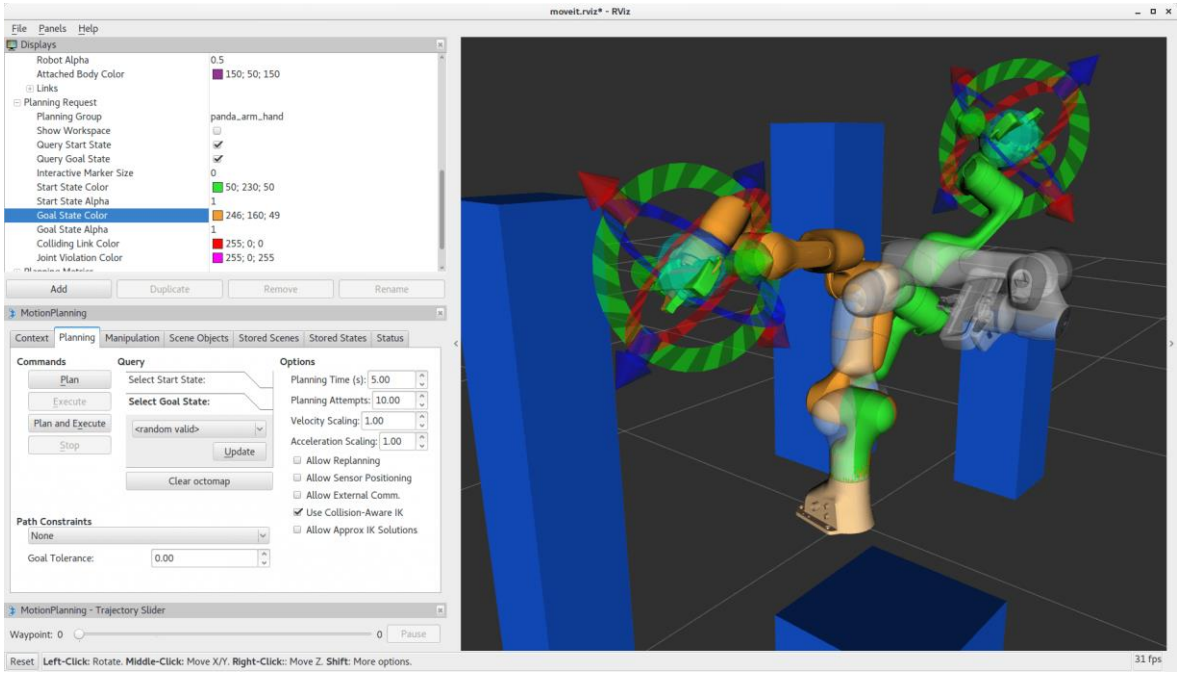
Robot model je definován jako XML soubor a obsahuje kinematický a dynamický popis, vizualizaci a kolizní model konkrétního robota. Na Obr. 15 je vidět část obsahující popis základny (base) našeho robota. Je zde definován souřadný systém, hmotnost a odkaz na .STL soubor obsahující 3D model tohoto linku.

```
<link name="Base">
        <inertial>
                <origin xyz="0.010476 0.000747 0.035226" rpy="0 0 0"/>
                <mass value="1.886"/>
                <inertia ixx="0.001196219" ixy="-0.000029358" ixz="0.000014859"
                                iyy="0.001147997" iyz="0.000016274"  izz="0.001425617"/>
        </inertial>
        <visual>
                <geometry>
                        <mesh filename="package://swiftpro/urdf/pro_links/Base.STL"/>
                </geometry>
                <origin xyz = "0 0 0 " rpy = "0 0 0"/>
                <material name = "">
                        <color rgba = "0.3 0.3 0.3 1" />
                </material>
    </visual>
</link>
```

Obr. 15 – Popis linku “Base” [6]

### 4.5.11. MoveIt

MoveIt je open-source knihovna sloužící k plánování pohybu robotů. Takto naplánované trajektorie mohou být vizualizovány pomocí nástrojů jako např. RViz. MoveIt dokáže pracovat s kolizním modelem robota a při plánování trajektorie testovat možné kolize a úspěšně se jim vyhýbat. Obsahuje také plugin MoveIt Setup Assistant, ve které v několika krocích definujete parametry svého robota jako např. popis linků, ze kterého je následně numerickou iterací možno vygenerovat rovnice inverzní kinematiky, např. pomocí pluginu IKFast a to až pro 6 stupňů volnosti.




Obr. 16 - Práce v MoveIt [23]

### 4.5.12. ROS Serial

ROS Serial je multiplatformní knihovna pro práci se sériovými porty. Poskytuje moderní rozhraní v C++ se všemi jeho výhodami, jako např. rychlost či ovladatelnost. [16] Použití ROS Serial protokolu dovoluje komunikaci s tzv. embedded systémy. Příkladem je Arduino.

# 5. Kinematický model

Kinematickým modelem rozumíme matematický popis mechanismu, ze kterého lze pro každý časový okamžik získat polohu, rychlost a zrychlení libovolného bodu. Mezi tři základní způsoby takového popisu se řadí: trigonometrické vyjádření, maticový způsob a vektorová metoda.

## 5.1. Trigonometrická metoda

Tato metoda spočívá ve vhodném dělení obrazce na trojúhelníky. Vhodné jsou zejména trojúhelníky, ve kterých se vyskytují hledané veličiny. Neexistuje však přesný návod, jelikož každá úloha je jiná a geometrické závisloti jsou různé. Tato metoda je vhodná pouze pro jednoduché rovinné mechanismy, pro složitější je velice nepřehledná a takřka nepoužitelná. [17]

## 5.2. Maticová metoda

Na rozdíl od trigonometrické metody je tento způsob kinematického popisu použitelný pro jakýkoliv mechanismus dle stejného schématu. Je vhodná pro řešení jak rovinných, tak i prostorových úloh. Vychází z maticového popisu současných pohybů, kdy každému pohybu přísluší vlastní transformační matice. Jsou to matice translace v osách x, y a matice rotace. Pronásobením těchto transformačních matic vznikne výsledná matice. Rychlosti a zrychlení částí mechanismu lze získat zderivováním těchto matic (pro rychlost je to první derivace polohy, pro zrychlení druhá derivace polohy). [17]

## 5.3. Vektorová metoda

Vektorová metoda spočívá v popisu rovinného mechanismu přes N uzavřených smyček, přičemž každá je skalárně rozepsána do souřadnic x a y:

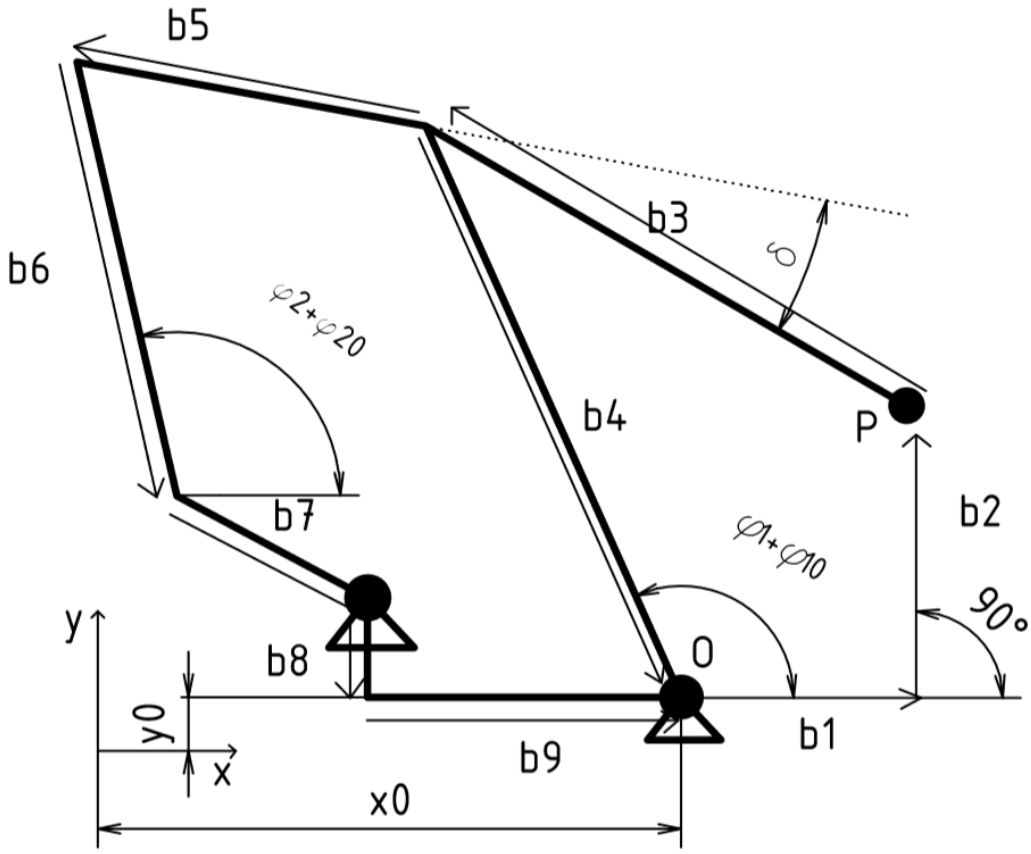
$$x: \sum_{i=1}^{m} b_i \cdot cos\beta_i = 0, \tag{5.1}$$

$$y: \sum_{i=1}^{m} b_i \cdot sin\beta_i = 0, \tag{5.2}$$

kde m je počet těles ve smyčce, $b_i$ je délka vektoru a $\beta_i$ je úhel vektoru od vodorovné osy x. Obdobně jako u maticového způsobu, zderivováním polohy dostaneme rychlost a zrychlení. Tento způsob je univerzální a lze jím popsat jakýkoliv rovinný mechanismus. [17] Tato metoda byla zvolena i pro popis kinematiky robota uArm Swift Pro.

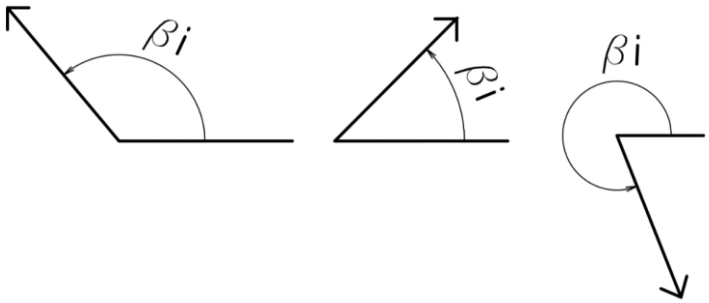
## 5.4. Vektorový popis první části mechanismu

Kinematický model první části mechanismu byl sestaven vektorovou metodou podle schématu na Obr. 17. Ta spočívá v popisu těles mechanismu pomocí skalárních rovnic rozepsaných do složek. V případě rovinného mechanismu jsou to složky x a y. Počet rovnic je závislý na počtu smyček. Kinematická struktura první části robota uArm Swift Pro má v rovině dvě smyčky, tzn. k popisu jsou potřeba čtyři rovnice.




Obr. 17 - Kinematické schéma první části rovinného mechanismu

Ve schématu nejsou pro přehlednost zobrazeny úhly $\beta_i$, jejich zavedení je však patrné z Obr. 18.




Obr. 18 - Zavedení úhlů beta

Na první pohled by se mohlo zdát, že motory jsou soustředné a tudíž v rovině xy leží v zákrytu, ovšem pro účely kalibrace je třeba podchytit i rozdíly v jejich umístění. Proto jsou v kinematickém schématu navíc vektory $b_8$ a $b_9$. Pokud by model sloužil pouze k vyjádření dopředné či inverzní kinematiky robota, mohla by se poloha pohonů považovat za shodnou. K další ilustraci slouží Obr. 19.


Obr. 19 - Kinematické schéma první části mechanismu na modelu

Rovnice vazeb budou tedy následující:

$$b_1 \cdot \cos(\beta_1) + b_2 \cdot \cos(\beta_2) + b_3 \cdot \cos(\beta_3) + b_4 \cdot \cos(\beta_4) = 0 \tag{5.3}$$

$$b_1 \cdot \sin(\beta_1) + b_2 \cdot \sin(\beta_2) + b_3 \cdot \sin(\beta_3) + b_4 \cdot \sin(\beta_4) = 0 \tag{5.4}$$

$$b_1 \cdot \cos(\beta_1) + b_2 \cdot \cos(\beta_2) + b_3 \cdot \cos(\beta_3) + b_5 \cdot \cos(\beta_5) + b_6 \cdot \cos(\beta_6) + \\ + b_7 \cdot \cos(\beta_7) + b_8 \cdot \cos(\beta_8) + b_9 \cdot \cos(\beta_9) = 0 \tag{5.5}$$

$$b_1 \cdot \sin(\beta_1) + b_2 \cdot \sin(\beta_2) + b_3 \cdot \sin(\beta_3) + b_5 \cdot \sin(\beta_5) + b_6 \cdot \sin(\beta_6) + \\ + b_7 \cdot \cos(\beta_7) + b_8 \cdot \sin(\beta_8) + b_9 \cdot \sin(\beta_9) = 0 \tag{5.6}$$

Rovnice (5.3) a (5.4) popisují smyčku mezi vektory 1, 2, 3, 4 v ose x a v ose y. Rovnice (5.5) a (5.6) popisují druhou smyčku tvořenou vektory 1, 2, 3, 5, 6, 7, 8, 9. Těmito čtyřmi rovnicemi je popsána kinematika první části robota uArm Swift Pro.

Ze schématu vyplývá, že úhly $\beta_1$, $\beta_2$, $\beta_8$, $\beta_9$ a $\delta$ jsou známé a konstantní pro každou polohu bodu P. Nezávislé proměnné jsou úhly natočení pohonů $\beta_4$ a $\beta_7$, pro které platí:

$$\beta_4 = \varphi_1 + \varphi_{10} + \pi, \tag{5.7}$$

$$\beta_7 = \varphi_2 + \varphi_{20} + \pi, \tag{5.8}$$

kde $\varphi_{10}$ a $\varphi_{20}$ jsou offsety měření.

Závislé proměnné jsou úhly $\beta_3, \beta_5$ a $\beta_6$, přičemž platí:

$$\beta_5 = \beta_3 - \delta. \tag{5.9}$$

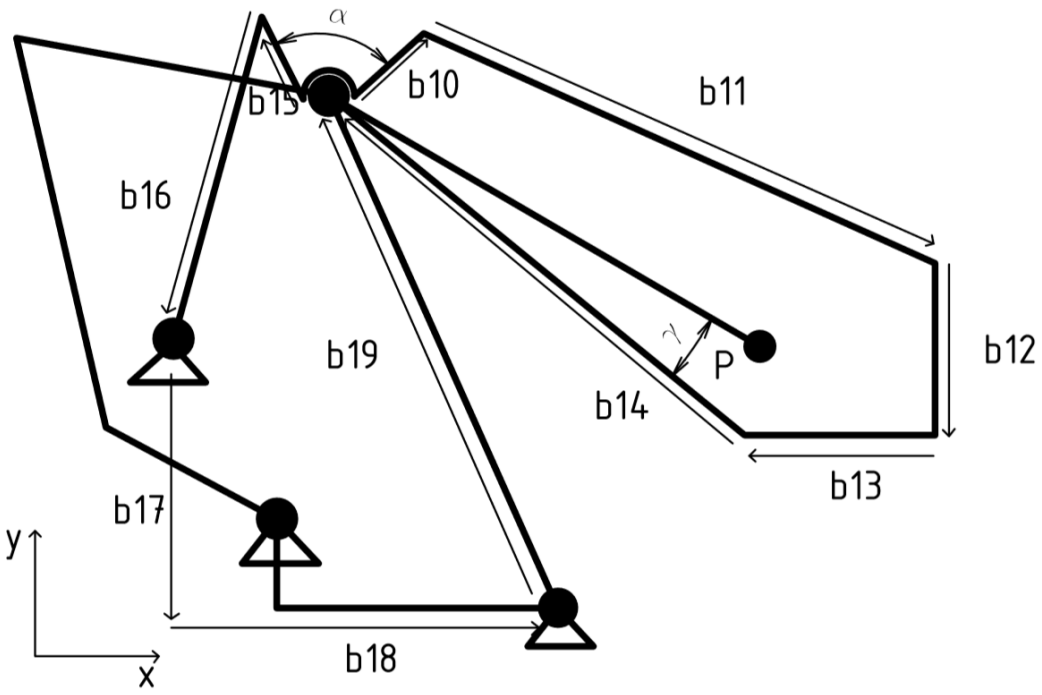
Dalšími závislými proměnnými jsou vzdálenosti $b_1$ *a* $b_2$ , ty je však možno vyjádřit jako

$$b_1 = x_P - x_0, \tag{5.10}$$

$$b_2 = y_P - y_0, \tag{5.11}$$

kde $x_P$ a $y_P$ jsou souřadnice pracovního bodu P a $x_0$ a $y_0$ jsou offsety popisující polohu robota vzhledem k počátku globálního souřadného systému.

## 5.5. Vektorový popis druhé části mechanismu




Obr. 20 – Kinematické schéma druhé části rovinného mechanismu

Druhá část mechanismu slouží pouze k udržení vodorovnosti koncového efektoru a je popsána dalšími dvěma smyčkami. Konstanty ve smyčkách jsou všechny délky $b_{10}, b_{11}, b_{12}, b_{13}, b_{14}, b_{15}, b_{16}, b_{17}, b_{18}, b_{19}$ a úhly $\beta_{17}$ a $\beta_{18}$. V každé poloze se mění úhly $\beta_{10}, \beta_{11}, \beta_{12}$ a $\beta_{16}$.

Pro vyjádření závislostí byly zavedeny konstantní úhly α a γ. Tato část mechanismu je tedy popsána rovnicemi:

$$b_{10} \cdot \cos(\beta_{10}) + b_{11} \cdot \cos(\beta_{11}) + b_{12} \cdot \cos(\beta_{12}) + b_{13} \cdot \cos(\beta_{13}) + b_{14} \cdot \cos(\beta_{14}) = 0, \quad (5.12)$$

$$b_{10} \cdot \sin(\beta_{10}) + b_{11} \cdot \sin(\beta_{11}) + b_{12} \cdot \sin(\beta_{12}) + b_{13} \cdot \sin(\beta_{13}) + b_{14} \cdot \sin(\beta_{14}) = 0, \quad (5.13)$$

$$b_{15} \cdot \cos(\beta_{15}) + b_{16} \cdot \cos(\beta_{16}) + b_{17} \cdot \cos(\beta_{17}) + b_{18} \cdot \cos(\beta_{18}) + b_{19} \cdot \cos(\beta_{19}) = 0, \quad (5.14)$$

$$b_{15} \cdot \sin(\beta_{15}) + b_{16} \cdot \sin(\beta_{16}) + b_{17} \cdot \sin(\beta_{17}) + b_{18} \cdot \sin(\beta_{18}) + b_{19} \cdot \sin(\beta_{19}) = 0, \quad (5.15)$$

přičemž platí následující:

$$\beta_{13} = \beta_{12} - \frac{\pi}{2}, \tag{5.16}$$

$$\beta_{14} = \beta_3 - \gamma, \tag{5.17}$$

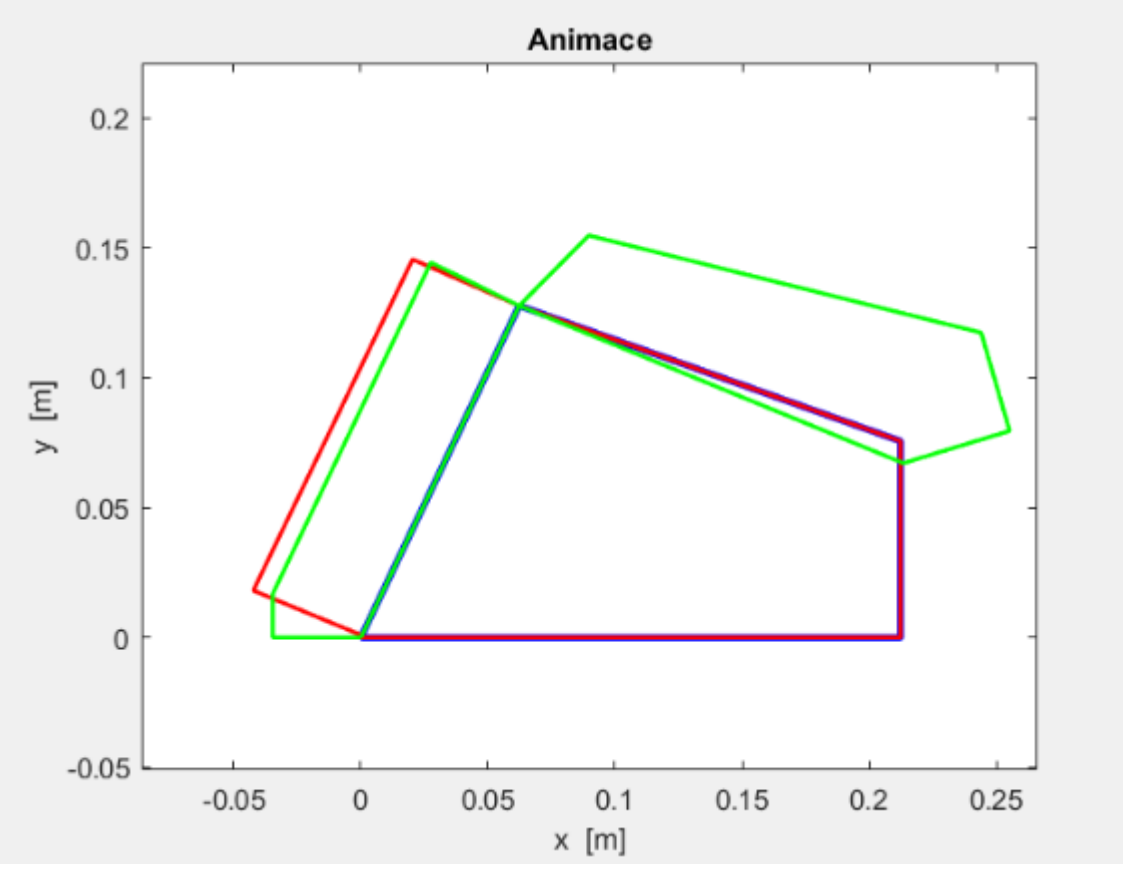
$$\beta_{15} = \beta_{10} + \alpha, \tag{5.18}$$

$$b_{19} = b_4, \tag{5.19}$$

$$\beta_{19} = \beta_4 - \pi. \tag{5.20}$$

Celkem je tedy mechanismus popsán čtyřmi smyčkami a osmi rovnicemi. V každé poloze je šest závislých proměnných a dvě nezávislé reprezentující pohony.

Programem Křešič bylo provedeno vykreslení pohybu mechanismu pro určitou trajektorii, čímž byla ověřena správnost kinematického popisu. Na Obr. 21 jsou znázorněny smyčky pro určitou polohu.




Obr. 21 - Animace pohybu mechanismu

# 6. Kinematická kalibrace

Kalibrován byl pouze rovinný mechanismus popsaný v kapitole 5. Pokud by měla být kalibrace provedena i pro svislou osu, zajišťující rotaci kolem základny, musel by být popis proveden např. pomocí maticové metody. Vektorová metoda není vhodná pro prostorové mechanismy, pro rovinné je však vhodná a plně dostačující.

## 6.1. Kalibrace poskytovaná výrobcem

Postup této kalibrace je následující. Po propojení robota s počítačem je potřeba zarovnat základnu robota s jeho siluetou na kalibrační šabloně, která je dostupná k vytištění na stránkách výrobce. Následně jsou příkazem *M2019* v prostředí Arduino deaktivovány motory, aby se dalo s rameny ručně hýbat. Na kalibrační šabloně jsou v přímce tři body: A, B, C. Koncový efektor robota je s těmito body postupně zarovnán a příkazy *M2401 A*, *M2401 B* a *M2401 C* se provede kalibrace. Ovšem tato „kalibrace" určí pouze offsety motorů, ne přesné rozměry délek ramen.

## 6.2. Kalibrační algoritmus

Základním pojmem je kinematická smyčka. Smyčkami jsou uzavřené obrazce uvnitř přidruženého grafu. Graf je sestaven na základě kinematického popisu robota. Každé těleso musí být obsaženo v alespoň jedné smyčce. Počet smyček se určí pomocí kostry přidruženého grafu, kde se redukuje počet propojení jednotlivých uzlů, dokud nevznikne pouze jedna uzavřená smyčka. Kinematické smyčky popisuje rovnice kinematických vazeb

$$\boldsymbol{f}(\boldsymbol{d}, \boldsymbol{s}, \boldsymbol{v}) = 0, \tag{6.1}$$

kde **d** zastupuje rozměry mechanismu, **s** naměřené hodnoty úhlů a délkových rozměrů a **v** jsou souřadnice koncového efektoru změřené externím měřidlem. Řešení probíhá přes modifikovanou Newtonovu metodu pro přeurčený systém nelineárních algebraických rovnic, splňujících vazbou rovnici (6.1). Označíme-li j=1,…, n počet změřených pozic, lze rovnici (6.1) přepsat do tvaru

$$\boldsymbol{F}(\boldsymbol{d}, \boldsymbol{S}, \boldsymbol{V}) = 0, \tag{6.2}$$

kde se každá j-tá pozice vyjádří z rovnice (6.1) jako

$$\boldsymbol{f_j}(\boldsymbol{d}, \boldsymbol{s_j}, \boldsymbol{v_j}) = 0, \tag{6.3}$$

a pro n pozic pišme

$$\boldsymbol{F} = [\boldsymbol{f_1}, \boldsymbol{f_2}, \dots, \boldsymbol{f_n}]^T, \tag{6.4}$$

$$\boldsymbol{S} = [\boldsymbol{s_1}, \boldsymbol{s_2}, \dots, \boldsymbol{s_n}]^T, \tag{6.5}$$

$$\boldsymbol{V} = [\boldsymbol{v_1}, \boldsymbol{v_2}, \dots, \boldsymbol{v_n}]^T. \tag{6.6}$$

Princip kalibrování je založen na tom, že rozměry **d** jsou stejné pro všechny změřené pozice. Nám jde o zjištění skutečných výrobních rozměrů, které jsou pro nás v tomto případě neznámé. Rozdíl mezi naměřenými a neznámými rozměry je však ve výrobní toleranci udávané výrobcem. Tyto výrobní rozměry označme $\bar{\boldsymbol{d}}$. Odvození Newtonovi metody vychází z Taylorova rozvoje rovnice (6.3)

$$\boldsymbol{F}(\bar{\boldsymbol{d}}, \boldsymbol{S}, \boldsymbol{V}) + \boldsymbol{J_d}\boldsymbol{\partial d} + \dots = 0, \tag{6.7}$$

kde $\boldsymbol{J_d}$ představuje Jakobiho matici parciálních derivací kinematických vazeb (6.3). Derivování probíhá podle kalibrovaných rozměrů **d**, z čehož vyplývá

$$\boldsymbol{J_d}\boldsymbol{\partial d} = -\boldsymbol{F}(\bar{\boldsymbol{d}}, \boldsymbol{S}, \boldsymbol{V}) = \boldsymbol{\partial r}, \tag{6.8}$$

a i-tá iterace Newtonovy metody je

$$\boldsymbol{\partial d_i} = (\boldsymbol{J_{di}^T J_{di}})^{-1} \boldsymbol{J_{di}^T} \boldsymbol{\partial r_i}, \tag{6.9}$$

kde $\boldsymbol{J_{di}}$ je Jakobiho matice a $\boldsymbol{\partial r_i} = -\boldsymbol{F}(\boldsymbol{d_i}, \boldsymbol{S}, \boldsymbol{V})$ je vektor odchylek počítaný z naměřených hodnot **S**, **V** a rozměrů mechanismu $\mathbf{d_i}$ z předchozího kroku iterace. Nově spočítaná hodnota rozměrů je

$$\boldsymbol{d_{i+1}} = \boldsymbol{d_i} + \boldsymbol{\partial d_i}. \tag{6.10}$$

Iterace probíhá, dokud se odchylky zmenšují. [18]

### 6.3. Data pro kalibraci

Původně bylo plánováno získání dat pro kalibraci měřením pracovního bodu P lasertrackerem, ovšem z důvodu nečekané poruchy zařízení a jeho následnému téměř dvouměsíčnímu odstavení bylo nutno data vygenerovat. Toho bylo dosaženo programem Křešič (viz příloha), který byl upraven pro dva nezávislé vstupy $\varphi_1$ a $\varphi_2$. Následně byly dopočítány odpovídající souřadnice $x_P$ a $y_P$ pracovního bodu P podle kinematického popisu uvedeného v kapitole 5. Dostali jsme tedy vektor souřadnic $Z = [\varphi_1, \varphi_2, x_p, y_p]^T$ pro každou polohu. Jelikož takto vypočítaná data neobsahují žádnou chybu měření a kalibrace by v tomto případě postrádala smysl, je třeba kalibrované hodnoty zatížit určitou chybou. Toho bylo docíleno mírnou změnou

přesných hodnot vstupujících do kalibračního algoritmu. K přesným hodnotám by měly po kalibraci dokonvergovat.

Také bylo pro účely kalibrace nutné nahradit délky $b_{12}$ a $b_{13}$ jednou délkou $b_{123}$, pro kterou platí vztah

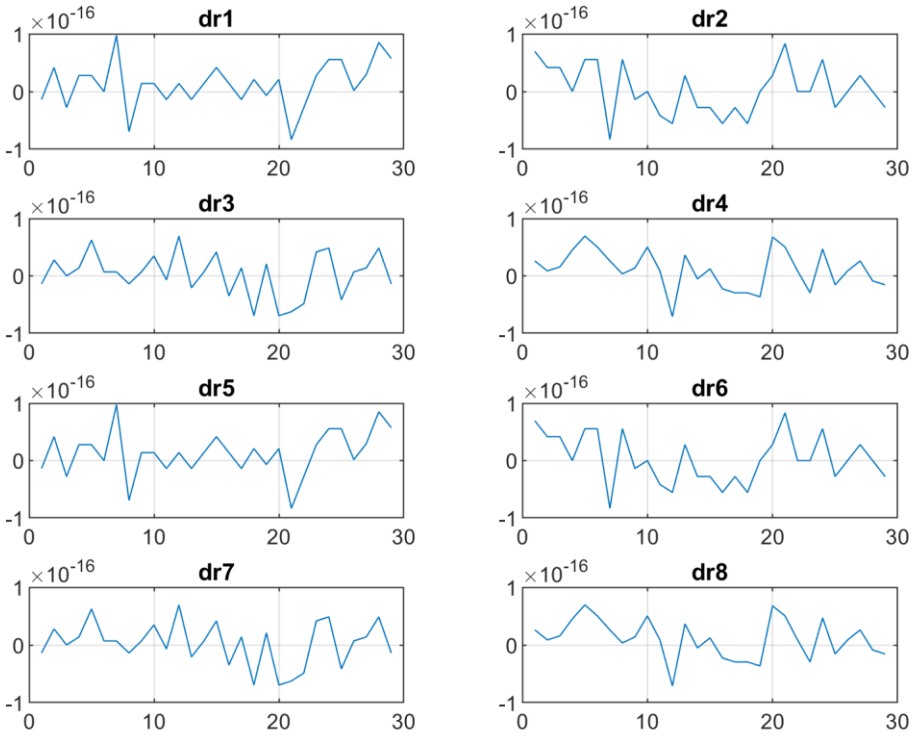
$$\overrightarrow{b_{123}} = \overrightarrow{b_{12}} + \overrightarrow{b_{13}}. \tag{6.11}$$

Toto nahrazení zamezí přeurčení soustavy. V případě použití dvou vektorů místo jednoho by totiž docházelo k neurčitosti a algoritmus by nedokázal efektivně zvolit tyto délky.

### 6.4. Kalibrační program

Program byl vytvořen v prostředí Matlab a funguje podle schématu popsaném v kapitole 6.2.

Kalibrace proběhla a podmínky byly splněny s přesností $10^{-16}$, což pro Matlab znamená numerickou nulu. To je patrné z grafu na Obr. 22, kde na ose x je i-tá poloha a na ose y odchylka od přesného řešení pro každou vazbovou rovnici.




Obr. 22 – Graf reziduí

Problémem je však vysoká kalibrovatelnost, řádově $10^{20}$. Kalibrovatelnost je vlastnost mechanismu být dobře zkalibrován a vypočítá se podle vztahu

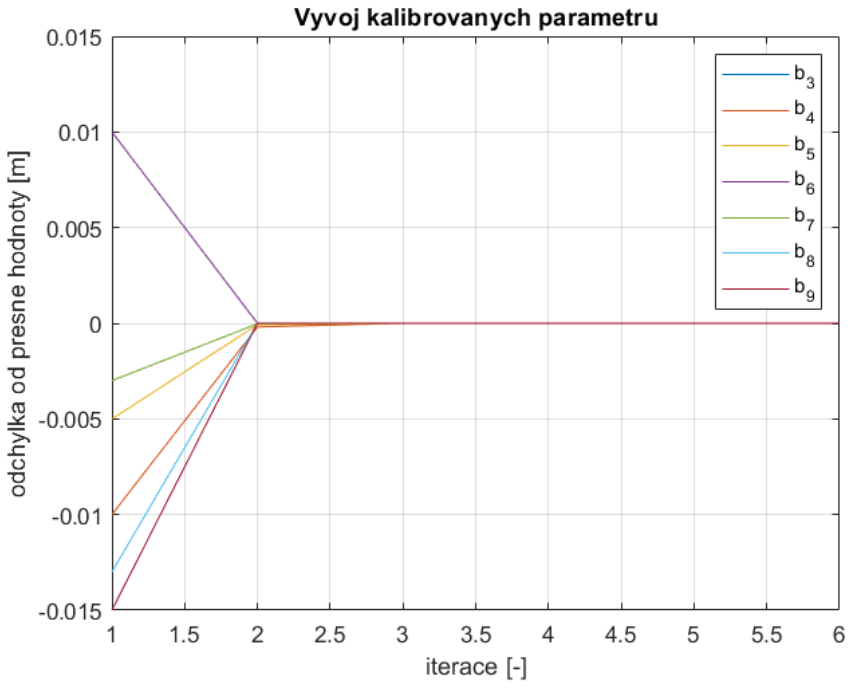
$$C = cond\left(\boldsymbol{J}_{\boldsymbol{d}}^{\boldsymbol{T}} \cdot \boldsymbol{J}_{\boldsymbol{d}}\right), \tag{6.12}$$

kde $\boldsymbol{J}_{\boldsymbol{d}}$ představuje Jakobiho matici parciálních derivací kinematických vazeb a funkce *cond* podmíněnost matice. Ideálně je kalibrovatelnost $C = 1$, může však nabývat hodnot $C \in \langle 1; \infty)$, přičemž za dobrou kalibrovatelnost reálného mechanismu se považuje maximálně $C = 10^{10}$ [19].

Tento problém je způsoben tím, že ve 3. ani 4. smyčce není žádná přesná délka. Jako závislé proměnné tam vstupují pouze úhly. To má za následek, že tyto smyčky i při splnění vazebních rovnic lze libovolně škálovat. Podrobnější analýzou pomocí singulárního rozkladu bylo zjištěno, že problém vzniká ve 3. smyčce a to konkrétně v úhlu $\beta_{15}$. Při splnění podmínek délky $b_{17}$, $b_{18}$ a $b_{15}$ konvergují k nule. Avšak $b_{16}$ splyne s $b_{19}$ a úhel $\beta_{15}$ může být prakticky libovolný a protože $\beta_{15} = \beta_{15} - \alpha$, problém se přenáší do 4. smyčky, a ta konverguje do jediného bodu, takže $b_{11}$, $b_{123}$ a $b_{14}$ se blíží nule.

Řešením by bylo buď měření délky libovolného vektoru ve 3. nebo 4. smyčce, nebo zavedení některého z vektorů jako přesný, nekalibrovaný rozměr. Tím by bylo udáno měřítko těchto smyček.

První ani druhé smyčky se tento problém netýká, neboť jsou jako závislé proměnné kromě úhlů i délky $b_1$ a $b_2$, takže měřítko je jasně dané.




Obr. 23 - Vývoj kalibrovaných parametrů první části mechanismu

Z Obr. 23 je vidět, že již po druhé iteraci je odchylka kalibrovaných rozměrů v první části rovinného mechanismu prakticky nulová od přesných hodnot.

# 7. Závěr

Hlavním cílem této práce bylo oživení robota uArm Swift Pro v prostředí ROS a následné naprogramování kinematické kalibrace rovinného mechanismu tohoto robota.

Prvním bodem bylo seznámení se s konstrukcí robotického ramena uArm Swift Pro a to jak se strukturovou, tak i se základní deskou Arduino MEGA 2560, jež se stará o řízení.

Následovalo nastudování si informací o historii, struktuře a programování v prostředí ROS. Základem bylo nalezení informací o základních komponentách ROSu a celkově filozofii práce v něm, která obnáší schopnost zacházení se strukturami jako ROS Nodes, ROS Messages, ROS Master a dalšími popsanými v kapitole 4.5. Po prvotním nastudování základů proběhla implementace knihoven pro robota uArm Swift Pro a vyzkoušel jsem si řízení v MoveIt a úspěšně otestoval komunikaci přes sériovou linku mezi ROSem a robotem při současné vizualizaci aktuálního stavu robota v prostředí RViz.

Třetím bodem bylo sestavení kinematického modelu robota (viz kapitola Kinematický model). Pro to jsem zvolil vektorou metodu. Robota lze rozdělit na dvě struktury. První část (viz kapitola Vektorový popis první části mechanismu), která obsahuje oba pohony a určuje polohu koncového efektoru a druhá část (viz kapitola Vektorový popis druhé části mechanismu), jež zodpovídá pouze za udržení rovnoběžnosti koncového efektoru s vodorovnou osou x.

Posledním bodem práce bylo seznámení se se základním schématem kalibrace, jejím matematickým odvozením a následná aplikace tohoto algoritmu na robota uArm Swift Pro.

Byly tedy splněny všechny cíle, které si tato práce stanovila.

# 8. Reference


[1] HWPRO, „www.hwpro.cz,“ 2019. [Online]. Available: https://www.robotshop.com/media/files/pdf/arduinomega2560datasheet.pdf. [Přístup získán 8 červen 2019].

[2] Ufactory, „www.ufactory.cc,“ [Online]. Available: http://download.ufactory.cc/docs/en/uArm%20Swift%20Pro_Developer%20Guide%20v1.0.6.pdf. [Přístup získán 5 červen 2019].

[3] B. Gerkey, „answers.ros.org,“ 6 prosinec 2011. [Online]. Available: https://answers.ros.org/question/12230/what-is-ros-exactly-middleware-framework-operating-system/. [Přístup získán 9 červenec 2019].

[4] Open Source Robotics Foundation, „wiki.ros.org,“ 21 červen 2014. [Online]. Available: http://wiki.ros.org/ROS/Concepts. [Přístup získán 20 červen 2019].

[5] P. Tomáš, Návrh a realizace senzorického systému pro mobilní robot s využitím frameworku ROS, Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, 2014.

[6] R. Cui a D. Long, „github.com,“ 2017. [Online]. Available: https://github.com/uArm-Developer/RosForSwiftAndSwiftPro. [Přístup získán 3 březen 2019].

[7] Open Source Robotics Foundation, „wiki.ros.org,“ 2018. [Online]. Available: http://wiki.ros.org/roslaunch. [Přístup získán 4 duben 2019].

[8] Open Source Robotics Foundation, „wiki.ros.org,“ 13 leden 2019. [Online]. Available: http://wiki.ros.org/msg. [Přístup získán 9 duben 2019].

[9] Open Source Robotics Foundation, „wiki.ros.org,“ 9 listopad 2018. [Online]. Available: http://wiki.ros.org/ROS/Tutorials/CreatingMsgAndSrv. [Přístup získán 23 duben 2019].

[10] Open Source Robotics Foundation, „wiki.ros.org,“ 18 červenec 2019. [Online]. Available: http://wiki.ros.org/Services. [Přístup získán 30 duben 2019].

[11] The MathWorks, Inc., „mathworks.com,“ 2019. [Online]. Available: https://www.mathworks.com/help/robotics/ug/ros-actions.html. [Přístup získán 30 duben 2019].

[12] Open Source Robotics Foundation, „wiki.ros.org,“ 14 duben 2019. [Online].

Available: http://wiki.ros.org/Packages. [Přístup získán 28 březen 2019].

[13] Open Source Robotics Foundation, „wiki.ros.org," 29 červen 2014. [Online]. Available: http://wiki.ros.org/rosparam. [Přístup získán 22 duben 2019].

[14] D. Hershberger, D. Gossow a J. Faust, „wiki.ros.org," 16 květen 2018. [Online]. Available: http://wiki.ros.org/rviz. [Přístup získán 15 březen 2019].

[15] Open Source Robotics Foundation, „gazebosim.org," 2019. [Online]. Available: http://gazebosim.org/. [Přístup získán 18 duben 2019].

[16] W. Woodall a J. Harrison, „wjwwood.io," 2012. [Online]. Available: http://wjwwood.io/serial/. [Přístup získán 5 duben 2019].

[17] M. Valášek, V. Bauma a Z. Šika, MECHANIKA B, Praha: Nakladatelství ČVUT, 2006.

[18] J. Krivošej, „Kinematická kalibrace zohledňující rozložení chyb čidel," České vysoké učení technické v Praze, Fakulta strojní, Praha, 2016.

[19] Z. Šika, V. Valášek a P. Beneš, „Calibrability as additional design criterion of parallel kinematic machines. Mechanism and Machine Theory," pp. 48-63, 2012.

[20] „hwpro.cz," [Online]. Available: https://www.hwpro.cz/oc/index.php?route=product/product&product_id=124.

[21] Ufactory, „www.ufactory.cc," červenec 2017. [Online]. Available: https://cdn.sparkfun.com/assets/3/d/3/a/8/uArm-Swift-Pro-Quick-Start-Guide.pdf. [Přístup získán 5 červen 2019].

[22] www.programmersought.com, „programmersought.com," 2019. [Online]. Available: http://www.programmersought.com/article/2135765036/. [Přístup získán 28 červen 2019].

[23] PickNic Consulting, „moveit.ros.org," 2019. [Online]. Available: https://moveit.ros.org/. [Přístup získán 5 duben 2019].

[24] S. Piperakis, „csd.uoc.gr," University of Crete, Computer Science Department, 21 březen 2019. [Online]. Available: https://www.csd.uoc.gr/~hy475/ROS-CS475.pdf. [Přístup získán 6 duben 2019].

[25] R. Chen, „medium.com," 22 červen 2018. [Online]. Available: https://medium.com/@raymonduchen/. [Přístup získán 28 červen 2019].