Jihočeská univerzita v Českých Budějovicích
Přírodovědecká fakulta

Konstrukce robotické ruky řízené pomocí ROS

Bakalářská práce

Daniel Hlavatý

Školitel: Mgr. Jiří Pech, Ph.D.

České Budějovice 2019

Bibliografické údaje:

Hlavatý, D., 2019: Konstrukce robotické ruky řízené pomocí ROS. [Construction of robotic arm controlled by ROS. Bc. Thesis, in Czech] – 87 p., Faculty of Science, University of South Bohemia, České Budějovice, Czech Republic.

Anotace:

Bakalářská práce se zabývá sestrojením modelu robotické ruky, který je řízený pomocí ROS a má částečně autonomní chování. Práce obsahuje výběr potřebného hardwaru a softwaru, popis konstrukčního řešení, výpočet kinematiky přímé i nepřímé a popis řízení pomocí ROS. Dále jsou zde rozebrány možnosti využití v praxi a je popsáno testování modelu.

Klíčová slova:

Robotika, sériový robot, Denavit-Hartenbergova konvence, ROS, Raspberry Pi, python

Annotation:

This theses describes the construction of robotic arm model that is controlled by ROS and has partly autonomous behaviour. The topics covered by the thesis are: selection of the necessary hardware and software, specification of construction, computing of forward and backward kinematics and description of control by ROS. Posibilities of practical usage are also described as well as the testing of the model.

Keywords:

Robotics, serial robot, Denavit-Hartenberg convention, ROS, Raspberry Pi, python

Prohlašuji, že svoji bakalářskou práci jsem vypracoval samostatně pouze s použitím pramenů a literatury uvedených v seznamu citované literatury.

Prohlašuji, že v souladu s § 47b zákona č. 111/1998 Sb. v platném znění souhlasím se zveřejněním své bakalářské práce, a to v nezkrácené podobě elektronickou cestou ve veřejně přístupné části databáze STAG provozované Jihočeskou univerzitou v Českých Budějovicích na jejích internetových stránkách, a to se zachováním mého autorského práva k odevzdanému textu této kvalifikační práce. Souhlasím dále s tím, aby toutéž elektronickou cestou byly v souladu uvedeným ustanovením zákona č. 111/1998 Sb. zveřejněny posudky školitele a oponentů práce i záznam o průběhu a výsledku obhajoby kvalifikační práce. Rovněž souhlasím s porovnáním textu mé kvalifikační práce s databází kvalifikačních prací Theses.cz provozovanou Národním registrem vysokoškolských kvalifikačních prací a systémem na odhalování plagiátů.

V Českých Budějovicích dne …… ……

Daniel Hlavatý

Poděkovaní

Rád bych poděkoval Mgr. Jiřímu Pechovi, Ph.D. za odborný dohled, cenné rady a čas věnovaný vedení této bakalářské práce. Také bych chtěl poděkovat své rodině a přítelkyni za podporu při mých studiích.

Obsah

1 Úvod

Lidé se již od středověku snaží vytvářet stroje, které by určitým způsobem nahrazovaly činnosti člověka. Od čistě mechanických záležitostí se naše společnost postupně dostala do bodu, kdy roboty dostávají lidskou podobu a vykazují vysokou míru inteligence zahrnující samostatného rozhodování a učení se. V dnešní době se pojem robot díky rychlému vývoji a neustále se rozšiřujícím možnostem využití objevuje stále častěji. Zatímco v minulosti byly robotické systémy spíše průmyslovým tématem, v současnosti se stávají nedílnou součástí běžných životů.

Robotické ruce jsou dnes neodmyslitelnou součástí průmyslové výroby. Zastávají širokou škálu pracovních činností jako zakládání dílů, manipulace s materiálem, montáže ve výrobních linkách, svařování nebo kontroly hotových dílů. Díky nim nemusí člověk vykonávat monotónní činnosti a snižuje se riziko vzniku úrazu při manipulaci během výrobního procesu.

ROS (Robot Operating System) je sada softwarových knihoven a nástrojů, sloužících pro tvorbu aplikací v oblasti robotiky. Poskytuje nové možnosti ve vývoji softwaru ovládajícího robotické systémy a je využíván jak pro servisní roboty, tak pro aplikace v průmyslové robotice. Tento systém je určen pro kohokoliv, ať už se jedná o velké průmyslové podniky, vědecká centra nebo pouze nadšence, kteří se robotice věnují ve volném čase.

Tato práce je zaměřena na zkonstruování robotické ruky, která vykazuje známky autonomního chování. Je zde popsána konstrukce vytvořeného modelu a řešeny otázky jeho polohování, popsáno elektrické zapojení komponent potřebných pro chod robotu a v neposlední řadě je rozebráno řízení sestrojeného modelu pomocí systému ROS.

2 Cíle práce

Hlavním cílem práce je zkonstruování modelu robotické ruky ovládaného pomocí ROS, který bude vykazovat známky autonomního chování. Dílčí cíle práce jsou:

- Popis možností řízení robotické ruky pomocí ROS.
- Sestrojení plně funkčního modelu robotické ruky s autonomním chováním řízeného pomocí ROS.
    - Rozhodnutí o operaci, kterou bude model vykonávat.
    - Konstrukce modelu.
    - Výběr vhodného hardwaru pro řízení pohybu robotické ruky.
    - Výběr vhodného softwaru pro řízení pohybu robotické ruky.
    - Naprogramování řízení robotické ruky.
- Popis konstrukce a zhodnocení možností využití v reálném světě.

3 Teoretická část

3.1 Robotika

Robotika je v dnešní době velmi rozšířeným oborem, zabývajícím se studiem robotů, jejich konstrukcí, designem, použitím a provozem. [1] Existuje několik definic jak robotiky, tak pojmu robot. Dosud však neexistuje ustálená definice, která by sjednotila názory odborníků tohoto oboru. Robot je všeobecně chápán jako stroj, který je schopen samostatně provádět podobné činnosti jako člověk. [2]

Robotika je interdisciplinární obor, který zahrnuje znalosti z mechaniky, elektroniky a elektrotechniky, automatizace, měřící techniky, informačních technologií, umělé inteligence a mnoha dalších vědních oborů. [1, 3]

Podle oblastí, kterými se robotika zabývá, ji můžeme rozdělit na:

- Teoretickou (robotika na úrovni základního případně aplikovaného výzkumu)
- Technickou (neboli robototechniku, na úrovni průmyslového výzkum a vývoje, zahrnuje vývoj jednotlivých částí robotů, řešení výpočtů, konstrukčních řešení apod.) [2]
- Aplikační (neboli robototechnologii, řešení nasazování průmyslových robotů do výrobních systémů, jejich programování a problémy s tím spojené) [2]

Podle aplikace lze roboty rozdělit na průmyslové a servisní. Servisní roboty nacházejí svoje uplatnění všude, kde se nejedená o výrobu produktů. Vykonávají širokou škálu servisních úloh (služeb) a částečně nebo zcela nahrazují činnosti běžně vykonávané lidmi. [4, 5]

3.1.1 Průmyslové roboty

Průmyslové roboty jsou roboty využívané v průmyslu, podílejí se tedy na výrobě produktů. Podle normy ISO 8373:2012 jsou definovány jako automaticky řízené, reprogramovatelné, víceúčelové manipulátory programovatelné ve třech nebo více osách, které mohou být fixovány na místě, nebo mobilní pro použití v průmyslové automatizaci. [6]

Průmyslové roboty jsou rozvinuté technické systémy, podílející se na výrobě především ve strojírenství, avšak nacházejí stále větší uplatnění i v jiných odvětvích. Základní vyráběné

koncepce průmyslových robotů jsou na vysoké technické úrovni a vynikají svou spolehlivostí, díky čemuž jsou nejrůznějšími výrobci neustále inovovány a jejich množství ve výrobě se stále zvyšuje. [3]

Podle typu kinematické struktury lze průmyslové roboty rozdělit na:

- Sériové (otevřený kinematický řetězec, výsledný pohyb složen z na sebe navazujících pohybů jednotlivých částí, které se mohou pohybovat nezávisle na sobě) [2, 7]
- Paralelní (uzavřený kinematický řetězec, výsledný pohyb je tvořen součinností všech částí zařazených vedle sebe, přičemž pohyby jedné části ovlivňují polohy ostatních) [2, 7]
- Kombinované (vznikají kombinací předchozích dvou struktur)

Obrázek 3.1: Paralelní robot IRB 360 FlexPicker společnosti ABB [8]

3.1.2 Struktura sériových robotů

Sériový robot (SR) je tvořen ze dvou částí. První z nich je nepohyblivá část, nazývaná základna (případně báze), která je pevně spojena s prostorem, ve kterém je realizován pohyb SR a určuje základní souřadný systém. Druhá část je hybná soustava, která je tvořena rameny propojenými klouby. Počet a druh kloubů určuje počet stupňů volnosti a rozsah pohybu robota.

Pohyb jednotlivých kloubů obstarávají aktuátory. Poslední člen hybné soustavy, který realizuje interakci s okolním prostředím, se nazývá koncový efektor (dále jen efektor). [7, 9]

Obrázek 3.2: Sériový robot IRB 2600 společnosti ABB [10]

SR představuje kinematický řetězec tvořený jednotlivými členy, které se vůči sobě vzájemně pohybují, čímž vznikají jednotlivé kinematické dvojice (vazby). V kinematických řetězcích průmyslových robotů se běžně vyskytují především dva druhy kinematických dvojic – rotační (R) a posuvné (T). Ty se indexují podle toho, ke které ose souřadného systému se jejich pohyb vztahuje. Kinematické řetězce bývají označovány jako posloupnost kloubů, reprezentující jednotlivé vazby, např.: RRR, RRT. [2]

3.1.2.1 Pracovní prostor

Pracovní prostor je množina všech bodů, které mohou být dosaženy efektorem. Je dán kinematickou strukturou robota a omezujícími podmínkami, jako např.: omezení maximálním vysunutím/natočením aktuátorů, nebo zamezení srážkám ramen. [9]

3.1.3 Kinematika SR

Kinematika zkoumá vzájemný pohyb částí kinematického řetězce, především pohyb koncového bodu (efektoru) vůči základně, v závislosti na proměnných kloubových

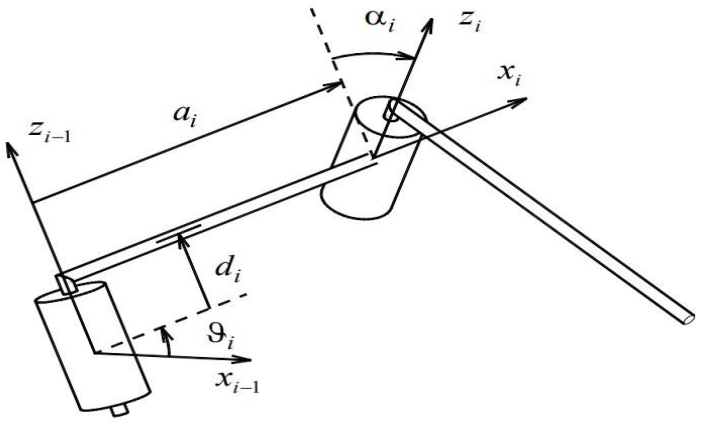
souřadnicích. Kloubové souřadnice představují natočení nebo posuny kinematických dvojic. S jednotlivými členy kinematického řetězce jsou pevně spojeny lokální kartézské souřadnicové systémy, přičemž se dále řeší jejich vzájemná poloha, a to buď maticovým, nebo vektorovým přístupem. Kinematiku jednodušších řetězců je možné řešit i použitím geometrických vztahů mezi jednotlivými členy. [1]

3.1.3.1 Denavit-Hartenbergovi parametry

Polohu a orientaci souřadných systému je možné definovat libovolně, ale v některých případech může být problematické sestavit transformační matice. V robotice se proto pro SR standardně používá Denavit-Hartenbergova konvence (dále D-H konvence), využívající čtyř parametrů, díky které se dají transformační matice sestavovat automaticky. [2, 11]

Princip transformace systémů je znázorněn na obrázku č. 3.3. Osy z jsou osami otáčení kloubů, případně osami ve směru posuvu. Transformační vztah systémů $x_{i-1}y_{i-1}z_{i-1}$ a $x_iy_iz_i$ je dán čtyřmi jednoduchými pohyby. Nejdříve natočíme osu x_{i-1} okolo osy z_{i-1} o úhel ϑ_i tak, že jsou osy x_{i-1} a x_i rovnoběžné. Poté posuneme osu x_{i-1} ve směru osy z_{i-1} o vzdálenost d_i tak, že jsou osy x_{i-1} a x_i totožné. Dále posuneme počátek souřadného systému $x_{i-1}y_{i-1}z_{i-1}$ ve směru osy x_i o vzdálenost a_i tak, že jsou totožné počátky systémů. Posledním krokem je natočit osu z_{i-1} okolo osy x_i o úhel α_i tak, aby souřadné systémy byly totožné. [2, 11]

Máme tedy čtyři parametry: ϑ_i, d_i, a_i a α_i, které jsou dále použity pro výpočty pomocí transformačních matic. [2, 11]



Obrázek 3.3: Denavit-Hartenbergův princip [2]

3.1.3.2 Přímá úloha kinematiky

Ze známých kloubových souřadnic a rozměrů SR hledáme kartézské souřadnice koncového bodu vzhledem k základnímu souřadnému systému. Její řešení pomocí geometrických vztahů

mezi jednotlivými členy, nebo pomocí lokálních souřadných systémů a výpočtů užitím transformačních matic není nijak složité a je vždy možné ji vyřešit analyticky. [2, 11]

3.1.3.3 Inverzní úloha kinematiky

V tomto případě jsou známé souřadnice koncového bodu a rozměry SR, ke kterým hledáme kloubové souřadnice. Inverzní úloha je tedy opakem úlohy přímé a její řešení je podstatně složitější, zvláště pro struktury s větším počtem stupňů volnosti. Ve většině případů je nemožné vyřešit úlohu analyticky a je zapotřebí sáhnout k iteračním numerickým metodám. [2, 11]

3.2 ROS (Robot Operating System)

Vytvoření komplexního robotického softwaru, který by mohl sloužit pro všeobecné účely a byl by určitým způsobem univerzální, zasahuje do široké škály oborů a předkládá mnoho úkolů a otázek, které je zapotřebí vyřešit. To je pro jednotlivce i pro jednotlivé instituce zabývající se robotikou takřka nemožné, a právě z toho důvodu byl vytvořen Robot Operating System. Ten poskytuje možnost spolupráce a společného budování robotických systémů, do kterých můžou jednotlivé skupiny, vynikající ve svých oborech, přispívat a navzájem na sobě stavět. [12]

ROS je flexibilní framework s otevřeným zdrojovým kódem, sloužící pro vytváření softwaru v oblasti robotiky. Je to sada nástrojů, knihoven a konvencí určená ke zjednodušení tvorby komplexních a robustních řídících programů pro širokou škálu robotických platforem. [12]

3.2.1 Struktura ROS

ROS se skládá z libovolného počtu nezávislých uzlů, které mezi sebou navzájem komunikují podle předem stanoveného modelu. Jednotlivé uzly nemusí být součástí jednoho systému ani stejné architektury, takže je možná mít jeden uzel v notebooku, další na Raspberry Pi a poslední v telefonu. Proto je ROS velmi pružným systémem, snadno přizpůsobitelným potřebám uživatele. [13, 14]

Systém komunikace mezi jednotlivými uzly je zajištěn následujícím způsobem. ROS Master pomůže uzlu nalézt ostatní uzly, se kterými poté může tento uzel komunikovat ohledně témat nebo služeb pomocí zasílání určitých typů komunikačních souborů. [15]

3.2.1.1 ROS Master

Master je uzel, který poskytuje jmenovací a registrační služby pro ostatní uzly v systému ROS a pomáhá uzlům ke vzájemné lokalizaci. Jakmile uzel najde jiný uzel, se kterým může navázat kontakt, začnou mezi sebou komunikovat peer-to-peer. [16]

Master bývá běžně spouštěn s dalšími dvěma uzly. První z nich je Parametr Server, což je sdílený vícerozměrný slovník, který využívají uzly pro ukládání a načítaní parametrů za běhu. Druhý je rosout, což je mechanismus hlášení protokolů, který zajišťuje posílání zpráv mezi uzly. [17]

3.2.1.2 ROS Nodes

Node (uzel) je naprogramovaný proces provádějící stanovenou funkci. Spojením uzlů vznikne celek, který je řídícím systémem pro určitou robotickou platformu. Jeden uzel může ovládat motory robotu, druhý může zajišťovat komunikaci s obsluhou, další uzly mohou mít na starost jednotlivá čidla a další mohou provádět výpočty potřebné pro správné chování robotického systému. [18]

Rozdělení celku v subsystémy s sebou přináší některé výhody. Vznik chyb a selhání je izolován na jednotlivé uzly a složitost kódu je ve srovnání s monolitickými systémy nižší. Jednotlivé uzly lze do celku přidávat a odebírat podle potřeby, a to bez větších zásahů do ostatních uzlů. Také detaily o implementaci jsou dobře skryté, protože uzly vystavují minimální API. [18]

Veškeré spuštěné uzly mají vlastní název, který je jednoznačně identifikuje pro zbytek systému. Kromě názvu mají uzly určený typ, což zjednodušuje proces odkazování na spustitelný uzel v souborovém systému (balíku). ROS poskytuje dvě dvojice uzlů, určené ke vzájemné komunikaci. [18]

Publisher a subscriber

Tyto dva uzly poskytují jednosměrnou komunikaci. Fungují tak, že publisher vysílá zprávy k danému tématu a subscriber má možnost se k tomuto tématu přihlásit a odesílaná data přijímat. [19]

Service a client

Service a client jsou uzly pro komunikaci se zpětnou vazbou. Client nejdříve vyhledá potřebnou službu a poté zašle daný typ souboru s požadavkem pro uzel service. Ten požadavek zpracuje a poté zašle uzlu client odpověď. [19, 20]

3.2.1.3 Topics a services

Jedná se o modely komunikace, kterými se mezi sebou vzájemně dorozumívají jednotlivé typy uzlů. Podle typu modelu zasílají nebo přijímají jednotlivé uzly zprávy vytvářené podle stanovené šablony.

Topics

Jak již bylo zmíněno, topics (témata) využívají uzly publisher a subscriber. Je to název pro komunikační spoje, přes které si uzly vyměňují zprávy. Publikování a přijímání informací je anonymně odděleno, takže uzly neví, s kým komunikují. K jednomu tématu muže vysílat data i více publisherů, stejně tak se k jednomu tématu může přihlásit více subscriberů. [19]

Services

Prostřednictvím services (služeb) komunikují uzly client a service. Služba je definována dvojicí zpráv, z nichž první je pro požadavek a druhá pro odpověď. Uzel service nabízí službu pod určitým názvem, kterou client zavolá odesláním požadavku a poté čeká na odpověď. Dalo by se říct, že se jedná o vzdálené volání procedury. [20]

3.2.1.4 Messages

Uzly komunikují pomocí messages (zpráv). Zpráva je jednoduchá datová struktura obsahující předtypované pole. Jsou podporovány primitivní datové typy (integer, string, bool atd.) případně pole těchto datových typů. [21]

msg files

Jsou to jednoduché textové soubory sloužící pro určení datové struktury, využívané pro generaci zdrojového kódu zprávy. Formát jazyka pro psaní těchto souborů je jednoduchý. Popis zprávy je seznam datových polí a definic konstant na samostatných řádcích. Soubory msg jsou využívány pro komunikace na určitá témata. [22]

srv files

Tento soubor v sobě obsahuje dvě zprávy, které jsou vzájemně oddělené. První z nich je požadavek, druhá je odpověď. Struktura jednotlivých zpráv je přímo založena na formátu souboru msg a slouží při komunikaci ohledně jednotlivých služeb. [23]

3.2.2 ROS Kinetic Kame

Vždy je dostupných více distribucí ROS, z nichž některé jsou podporované po delší časové období (LTS distribuce), což je dělá stabilnějšími. LTS distribuce jsou podporované po dobu pěti let, zatímco krátkodobé pouze po dobu let dvou. Od roku 2015 je uvolněna nová distribuce každý rok v květnu a střídá se dlouhodobá s krátkodobou. [24, 25]

Distribuce ROS je sada balíčků ROS o určité verzi, které jsou podobné distribucím Linuxu. Vývojáři mají tedy možnost pracovat s relativně stabilní databází, protože po uvolnění distribuce je snaha provádět jen minimální zásahy do balíčků (opravy chyb apod.). [24]

ROS Kinetic Kame je tedy jednou z distribucí ROS, která byla vydána 23. května 2016 a její podpora končí v dubnu roku 2021. Primárně je určená pro Ubuntu 16.04, ale je v určité míře kompatibilní i s jinými linuxovými systémy, Mac OS X, Android či Windows. [24, 25]

4 Použitý hardware

Tato kapitola se zabývá hardwarem, který byl použit pro sestrojený model robotické ruky. Výběr komponent byl ve většině případů jednoduchý, hlavním aspektem byla vhodnost jejich použití pro stanovený účel.

4.1 Raspberry Pi 3 Model B

Raspberry Pi je plnohodnotný počítač, který se svými rozměry může rovnat velikosti kreditní karty. Vyrábí se v několika variantách a použitá verze, Model B třetí generace, byla vydána v únoru roku 2016. Zvolený model nabízí 64–bitový čtyřjádrový procesor ARM Cortex-A53 pracující na frekvenci 1,2 GHz. Procesor má k dispozici 1 GB operační paměti, která je sdílená s grafickým procesorem Broadcom VideoCore IV. [26] Poskytuje standardní škálu vstupů a výstupů jako jsou HDMI, USB 2.0 porty, 3,5 mm jack, konektor RJ-45 nebo microSD slot pro možnost vložení interní paměti a nabízí podporu Wi-Fi a Bluetooth. Dále se na desce nachází GPIO rozhraní se 40 piny, ke kterým je možné připojovat další hardware, např.: senzory, LED diody, nebo jednotlivé moduly přímo vyráběné pro Raspberry Pi. [27]

Raspberry Pi bylo vybráno z několika důvodů. Pro ROS byl vybrán operační systém Ubuntu, který je jednou z distribucí Linuxu a pro své spuštění potřebuje z důvodu vzdáleného přihlašování přístup k internetu. Pro řízení robotické ruky je potřeba ovládat aktuátory a zároveň číst údaje z čidel a senzorů. Na Raspberry Pi je možné provozovat distribuce Linuxu, jsou zde dvě možnosti pro přístup k internetu a GPIO rozhraní, ke kterému je možné připojovat potřebné komponenty.

K chodu samotného Raspberry Pi a běhu operačního systému byla pořízena paměťová karta typu microSDHC od výrobce Kingston s kapacitou 32 GB. Také byl vybrán 3 A zdroj, který poskytuje dostatečný výkon pro připojení základních periferií.

4.2 Aktuátory

4.2.1 Motor SX17-1005VLQCEF

Jedná se o hybridní dvoufázový krokový motor výrobce Microcon v přírubě Nema 17, která má rozměry 42,3 x 42,3 x 40 mm. Tento motor má vinutí zapojená bipolárně do série a na výstupním hřídeli poskytuje statický moment 0,52 N·m při jmenovitém proudu 1 A. Na jednu

otáčku připadá 200 kroků, délka jednoho kroku je tedy 1,8° a krok je možné elektronickým řízením dále rozdělit. Pro řízení krokového motoru byl zvolen ovladač A4988. [28]

Tento motor byl vybrán pro realizaci translačního pohybu z důvodu dostatečné rezervy kroutícího momentu a vyhovujícím rozměrům.

4.2.1.1 Ovladač A4988

A4988 je ovladač pro bipolární krokové motory s vestavěným překladačem pro jednoduché řízení. Má nastavitelnou hodnotu proudu do motoru, je kompatibilní s logickým napětím 3,3 a 5 V a je chráněn proti zkratu, přehřátí a podpětí. Výstupní kapacita pro motor je 12–36 V a až ± 2 A. Ovladač může pracovat v režimu pro plný krok, nebo v režimech mikrokrokování s nejvyšším rozlišením 16 mikrokroků na plný krok. Tento ovladač byl vybrán z důvodu kompatibility se zvoleným motorem a Raspberry Pi a pro jeho nízkou cenu. [29]

4.2.2 Motor 28BYJ-48 5VDC

Tento malý unipolární krokový motor opatřený převodovkou poskytuje statický moment 0,03 N·m, při vstupním napětí 5 V. Motor má průměr 28 mm, výšku 19 mm a jeho hmotnost se pohybuje okolo 40 g. Mechanický krok motoru je 5,625°, avšak výsledný krok na hřídeli, zredukovaný převodovkou v poměru 1/64, má pouhých 0,08789°. Motor je běžně prodáván s ovladačem obsahujícím čip UNL2003A, který je pro řízení tohoto motoru dostačující. [30]

Tyto motory byly použity dva, a to jako pohony pro rotační pohyby ramen modelu. Zvoleny byly z důvodu malých rozměrů, nízké hmotnosti, krátkého kroku pro možnost celkem přesně řídit polohu a také pro jejich nízkou cenu.

4.2.3 Servomotor GH-S43A

Miniaturní modelářský servomotor, vážící pouhých 4,3 g, pracuje při provozním napětí v rozmezí 4,8–7,2 V. Díky nízké hmotnosti a malým rozměrům je často používán pro aeromodeling. Při 4,8 V se zvládne motor pootočit o 60° za 0,1 s a poskytuje kroutící moment 0,06 N·m. Rozsah pootočení servomotoru je přibližně 230° a díky PWM regulaci (pulzně šířková modulace) je řízení natočení výstupního hřídele vcelku přesné. [31]

Servomotor je použit pro ovládání koncového efektoru. Byl zvolen díky malým rozměrům, nízké hmotnosti a dostačujícímu kroutícímu momentu. PWM regulace je řešená pomocí Raspberry Pi, které má v GPIO rozhraní několik pinů, poskytujících tento druh regulace.

4.3 Senzory a čidla

4.3.1 Infračervený senzor překážek FC-51

Senzor má dvojici diod, z nichž jedna je vysílačem infračerveného záření o určité frekvenci a druhá z nich je naopak přijímačem. Senzor funguje tak, že vysílač neustále vysílá záření a v momentě, kdy se před senzorem objeví předmět, se záření odrazí zpět k senzoru a dopadá na přijímací diodu. Signál vznikající na přijímači je dále zpracován obvodem senzoru a na výstupním pinu senzoru se objeví logická úroveň 0 (nízká úroveň). Rozsah vzdálenosti detekce je nastavitelný trimrem v rozmezí 2-15 cm. Pracovní napětí senzoru je od 3,3 do 5 V. [32]

Senzor slouží k detekci předmětu na pozici a kontrole uchopení předmětu. Byl zvolen z důvodu jeho jednoduchého použití, kompatibilitě s Raspberry Pi, nízké ceně a také proto, že na funkci senzoru nemají zásadní vliv změny osvětlení v jeho okolí. Jedinou nevýhodou tohoto senzoru je pohlcování infračerveného záření černou barvou, takže detekované předměty nemohou být této barvy.

4.3.2 Ultrazvukový senzor HC-SR04

Tento senzor poskytuje měření díky vysílači a přijímači ultrazvukového vlnění. Senzor funguje tak, že po obdržení spouštěcího signálu (vysoká úroveň po dobu 10 μs) vyšle vysílač měřící signál, který po odražení od objektu před senzorem dopadá na přijímač. Dopadající signál je zpracován obvodem a výstupem měření je vysoká úroveň na výstupním pinu senzoru po určitou dobu. Z této doby se poté pomocí jednoduchého výpočtu získá vzdálenost mezi senzorem a objektem. Senzor použitý pro model má měřící rozsah 5-400 cm s nepřesností ± 3 mm a jeho provozní napětí je 5 V. [33]

Senzor slouží jako přibližná kontrola pro translační pohyb. Hlídá, zda nedošlo ke ztrátě kroku motoru. V původním návrhu senzor nebyl, k přidání došlo později kvůli zvýšení množství vstupů a tím pádem přidání uzlu v ROS pro jeho otestování. Vybrán byl pro jednoduchost zapojení a nízkou cenu.

4.3.3 Koncové spínače

Pro zadanou aplikaci je potřeba najet robotem do základní polohy, od které se budou odvíjet výpočty pro další polohy. Najetí jednotlivých ramen do jejich základních pozic je zajištěno pomocí mechanických koncových spínačů. Ty fungují na jednoduchém principu, kdy po

stlačení páky dojde k mechanickému rozepnutí obvodu. V tu chvíli se na výstupu spínače objeví logická úroveň 0 (nízká úroveň).

Spínače nebyly vybírány podle žádných specifických kritérií, proto byl zvolen spínač s vyhovujícími rozměry pro jednoduchou montáž a s dostatečně dlouhým modulárním kabelem.

4.4 Napájení

Pro napájení aktuátorů je použit externí zdroj, aby vyšší hodnoty proudu z výkonových částí obvodů netekly skrz Raspberry Pi, pro které by taková zátěž mohla být destruktivní. Jednotlivé motory je potřeba napájet požadovaným pracovním napětím, což je vyřešeno použitím snižovacích a zvyšovacích měničů z důvodu, že jejich použití je cenově výhodnější než použití více zdrojů.

4.4.1 Spínavý zdroj LRS-50-12

O dodání energie pro aktuátory se stará spínaný zdroj LRS-50-12 od výrobce Mean Well o výkonu 50,4 W. Ten transformuje napětí ze sítě na napětí výstupní, které je regulovatelné v rozmezí 10,2-13,8 V. Zdroj je při výstupním napětí 12 V schopen dodat proud 4,2 A a jeho účinnost je 86 %. Zdroj má integrované ochrany proti přetížení, přepětí a přehřátí. [34]

Pro výběr zdroje byla rozhodující dostatečná hodnota proudu, kterou dokáže poskytnout. Zdroj byl také vybírán s dostatečnou rezervou pro případné nástavby sestrojeného modelu. Dále bylo při výběru přihlédnuto k rozměrům a ceně zdroje.

4.4.2 DC-DC XL6009 step up/down modul

Tento modul je často používán ve fotovoltaice, jelikož dokáže stabilizovat výstupní napětí na určitou hodnou bez ohledu na to, jestli je vstupní napětí nižší nebo vyšší než napětí výstupní. Měnič má vstupní napětí v rozsahu 5-32 V a výstupní napětí je regulovatelné trimrem v rozmezí 1,25-35 V. Účinnost transformace napětí je 93,5 %, výstupní proud může být až 2 A a výstupní napětí je zvlněné maximálně o hodnotu ±50 mV. [35]

Modul je použit pro zvýšení napětí ze zdroje (12 V) na 24 V pro napájení motoru SX17-1005VLQCEF. Při výběru bylo zásadním parametrem maximální proudové zatížení měniče a možnost zvýšit napětí na 24 V. Dále bylo přihlédnuto k možnosti přichycení modulu pomocí montážních otvorů.

4.4.3 DC-DC step-down měnič s LM2596

Tento měnič slouží ke snižování vstupního napětí. Vstupní napětí měniče se pohybuje v rozsahu 4,5-40 V a výstupní napětí je regulovatelné trimrem v rozmezí 1,2-37 V. Účinnost transformace se při hodnotě výstupního napětí 5 V pohybuje okolo 80 %. [36]

Měnič je použit pro snížení napětí ze zdroje (12 V) na 5 V pro napájení dvou motorů 28BYJ-48 a servomotoru GH-S43A. Při výběru bylo zásadním parametrem maximální proudové zatížení měniče a možnost snížit napětí na 5 V. Dále bylo přihlédnuto k možnosti přichycení modulu pomocí montážních otvorů.

4.5 Ostatní komponenty

4.5.1 Tlačítko

Pro předání informace, že je potřeba provést stanovený úkon, bylo zvoleno ovládání pomocí jednoho tlačítka. Tlačítko funguje tak, že po stisku dojde k rozepnutí obvodu a na výstupu se tedy objeví logická úroveň 0. Tlačítko bylo vybíráno tak, aby se dalo umístit do ovládacího prvku určeného obsluze robotické ruky.

4.5.2 RGB LED modul KY-16

Pro indikaci stavu robota obsluze bylo zvoleno barevné rozlišení pomocí diod. Tento modul má však jen jednu diodu, která je složená ze tří samostatných diod, které mají společnou katodu a poskytují tři základní barvy (červená, modrá a zelená). Modul byl zvolen právě kvůli integraci všech tří barev do jediné součástky.

5 Použitý software

V této kapitole bude probrán software, použitý pro realizaci pohybu robotické ruky. ROS byl probrán v teoretické části, proto je zbytečné ho znovu uvádět v této kapitole.

5.1 Ubuntu

Ubuntu je softwarová platforma určená například pro počítače, servery, telefony nebo zařízení spadající do kategorie internetu věcí. Ubuntu je bezplatnou linuxovou distribucí založenou na Debianu, která je vyvíjená společností Canonical. [37]

5.1.1 Ubuntu Core 16

Ubuntu Core je operační systém, který je minimalistickou verzí klasického Ubuntu. Systém je vysoce zabezpečený a balíčky jsou aktualizovány vzdáleně. Ubuntu Core bylo vytvořeno pro potřeby internetu věcí a Canonical poskytuje obrazy tohoto operačního systému pro nejvíce rozšířené platformy v této oblasti, mezi které patří i Raspberry Pi. Ubuntu Core 16 je dlouhodobě podporovanou verzí Ubuntu 16.04, která vyšla v dubnu roku 2016. [38]

Jak již bylo zmíněno, ROS Kinetic Kame je primárně určen pro Ubuntu 16.04. Volba hostujícího systému pro ROS byla tedy jednoznačná a pro Raspberry Pi je použit operační systém Ubuntu Core 16.

5.2 Python

Python je vysoce výkonný vysokoúrovňový programovací jazyk, který vytvořil na začátku devadesátých let minulého století Guido van Rossum. Jedná se o projekt s otevřeným zdrojovým kódem, jehož interpret je dostupný pro běžně používané platformy. Python nabízí dynamickou kontrolu datových typů a automatickou správu paměti. Je jednoduchý na naučení a díky dobré syntaxi je snadno čitelný. [39-41]

Pro psaní zdrojového kódu programů v ROS je možné využít programovacího jazyka C, nebo jazyk Python. Pro Raspberry Pi je Python doporučovaný vývojový jazyk (druhé slovo v jeho názvu vzniklo zkomolením jména Python) [42] a Ubuntu jako linuxová distribuce ho podporuje také. Z těchto důvodů byl pro psaní jednotlivý uzlů použit právě Python.

5.3 Použité moduly

5.3.1 RPi.GPIO

RPi.GPIO je modul zpřístupňující rozhraní GPIO. Obsahuje funkce pro ovládání a čtení základních pinů a složitější funkce pro ovládání speciálních pinů v rozhraní, jako jsou například piny nabízející PWM. [43]

5.3.2 math

Tento modul je v Pythonu vždy dostupný a poskytuje přístup k matematickým funkcím definovaným standardem C. Tyto funkce však nelze použít s komplexními čísly, pro ty by bylo zapotřebí importovat modul cmath. [44]

5.3.3 time

Modul time poskytuje nejrůznější funkce spojené s časem. Na některých platformách však nejsou dostupné veškeré funkce, jelikož se jednotlivé platformy mezi sebou mohou lišit funkcemi obsaženými v knihovně C, kterou modul time volá. [45]

6 Praktická část

Praktická část této práce je rozdělena do tří úseků. První z nich se zabývá konstrukčním řešením robotické ruky, sestavením navrženého modelu, potřebnými výpočty pro určení polohy koncového efektoru a natočení aktuátorů. Ve druhém je rozebráno zapojení jednotlivých komponent potřebných pro správnou funkci ruky. Ve třetím je popsána instalace potřebného softwaru na Raspberry Pi a je zde vysvětleno řízení modelu pomocí ROS. Dále jsou zde rozebrány zásadní části zdrojových kódů, které model řídí.

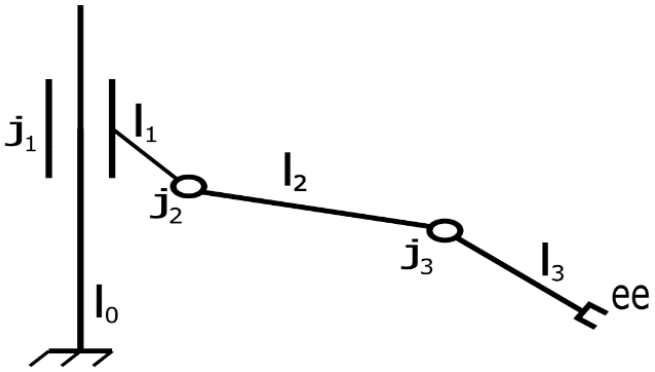
Autor se při výběru prováděné činnosti rozhodl pro jednu ze základních operací sériových robotů, kterou je manipulace s předmětem. Autonomní chování bude představovat kontrolování jednotlivých pozic během chodu programu a následné vyhodnocování zjištěných stavů.

6.1 Mechanická konstrukce

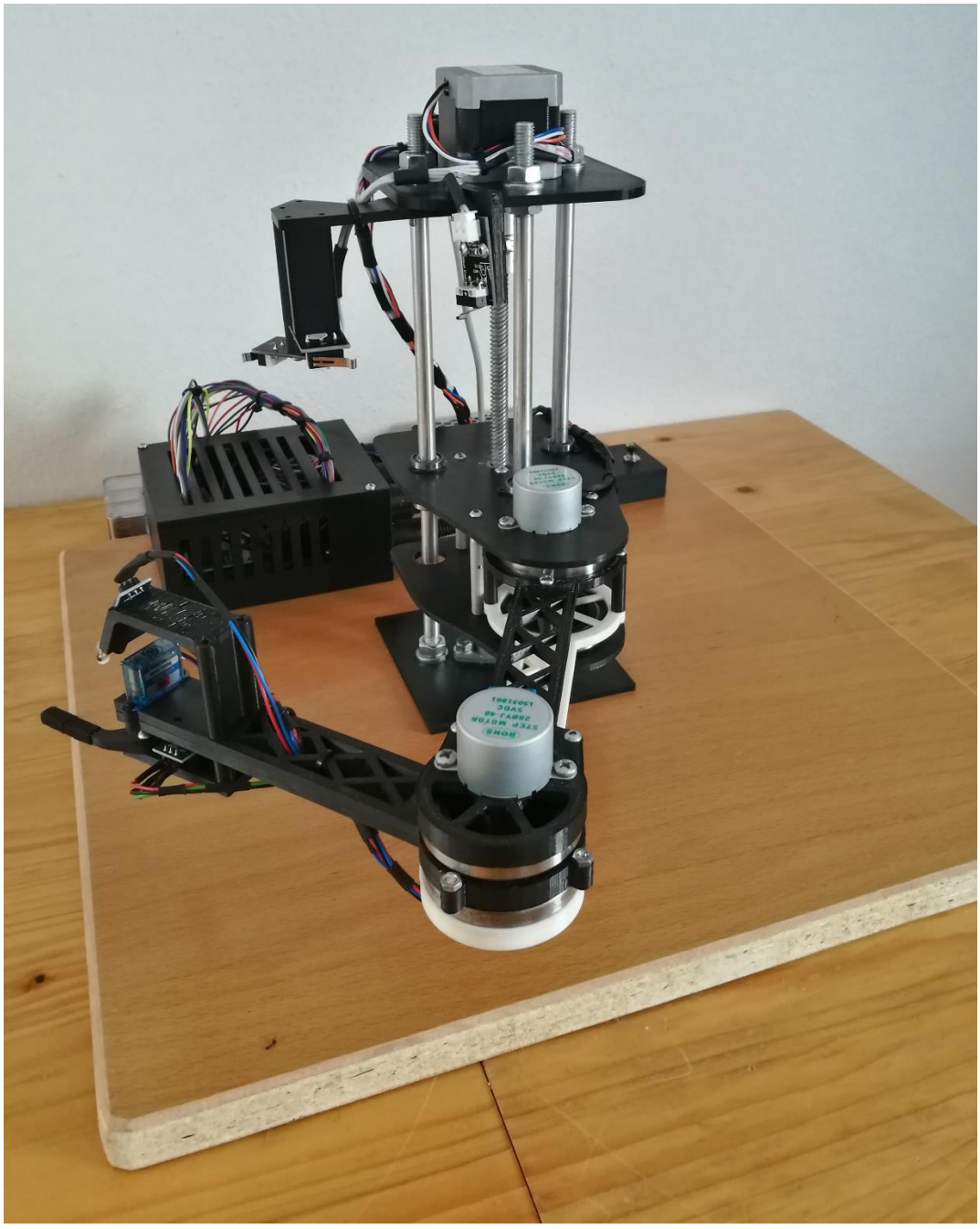
Konstrukce modelu byla řešena s ohledem na technologie dostupné autorovi a pro návrh dílů byl použit program Solidworks. Možnost nechat si vyrobit potřebné díly z kovu pro autora byla finančně nedosažitelná, proto byla pro výrobu dílů využita technologie 3D tisk, která se v posledních letech velice rozšiřuje. Zkonstruované díly proto musely být navrženy tak, aby bylo možné je tisknout bez velkých obtíží. Díly jsou vytištěny z plastu, konkrétně z materiálu PET (polyethylentereftalát), takže mají horší mechanické vlastnosti, než by měly díly z kovu. Proto byly vybírány komponenty, především tedy aktuátory tak, aby konstrukce nebyla příliš namáhána a nedocházelo k velkým deformacím. Fotografie kompletního modelu je k vidění na obrázku č. 6.2.

6.1.1 Návrh kinematické struktury

Pro jednoduchost výpočtů a jednoduché konstrukční řešení byl zvolen kinematický řetězec $T_zR_zR_z$, který má tři stupně volnosti. Jedná se o obrácený kinematický řetězec v praxi často využívaného robotu typu SCARA ($R_zR_zT_z$). Řešení translačního pohybu by v případě modelu založeného na řetězci typu SCARA zatěžovalo plastová ramena příliš vysokým ohybovým namáháním, proto byl použit řetězec obrácený. Kinematický řetězec modelu je na obrázku č. 6.1. Členy jsou očíslovány podle D-H konvence, člen l_0 je základna, členy l_n jsou ramena a členy j_n jsou pohybové jednotky (klouby). Poslední rameno je zakončeno efektorem (ee).



Obrázek 6.1: Kinematický řetězec modelu



Obrázek 6.2: Sestrojený model robotické ruky

6.1.2 Realizace pohybů

Realizace rotačních pohybů je vzhledem k použití krokových motorů 28BYJ-48 velice jednoduchá. Motory vykonávají rotační pohyb, který je pomocí výstupního hřídele přenášen přímo na rameno. Přenos je zajištěn dírou v rameni, která rozměry a tvarem odpovídá výstupnímu hřídeli motoru. Ramena jsou uložena v ložiskách 6807-2RS o rozměrech 47 x 35 x 7 mm.

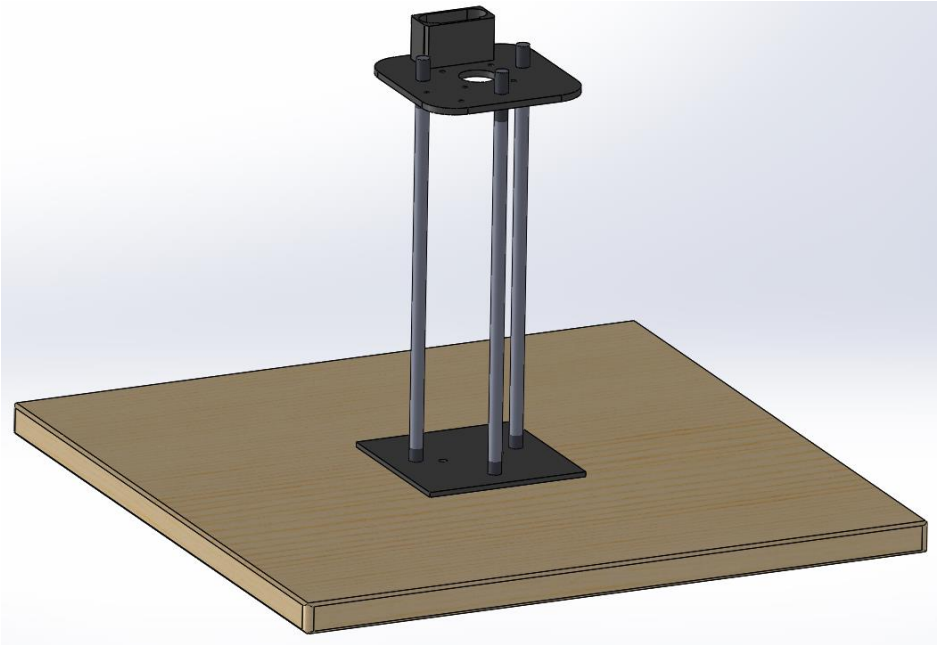
Pro řešení translačního pohybu bylo po prostudování dostupných možností použito převodu rotačního pohybu krokového motoru SX17-1005VLQCEF na translační pomocí pohybového šroubu. Pro model je použit šroub trapézový, protože je díky tření v závitech samosvorný a není tedy pro zamezení samovolného pohybu potřeba použít brzdu, jako v případě šroubu kuličkového. Použitý šroub má průměr 8 mm, jeho délka je 200 mm a jeho stoupání je 8 mm na otáčku. Šroub je spojen s hřídelem motoru pomocí pružné spojky a na druhém konci je uložen v ložiskovém tělese, ve kterém je zajištěn pomocí dvou červíků. Lineární vedení je tvořeno třemi vodícími tyčemi o průměru 8 mm, na kterých se pohybují lineární ložiska upevněná v rameni, kterým je pohybováno.

6.1.3 Konstrukce

6.1.3.1 Základna

Základna byla navržena jako dřevěná deska o rozměrech 40 x 40 x 1,9 mm spojená s lineárním vedením, které je ukončené vrchním dílem. Dřevěná deska je krom vedení spojena s menším plastovým dílem, sloužícím pro správné umístění otvorů ve dřevěné desce. Jak již bylo popsáno, vedení je tvořeno třemi tyčemi a každá z nich je na obou koncích opatřena závity pro realizaci šroubových spojů. Jedním koncem jsou tyče maticemi upevněny k desce, na druhém konci je maticemi připevněn vrchní díl, ke kterému je uchycen motor SX17-1005VLQCEF, zajišťující pohyb prvního ramena.

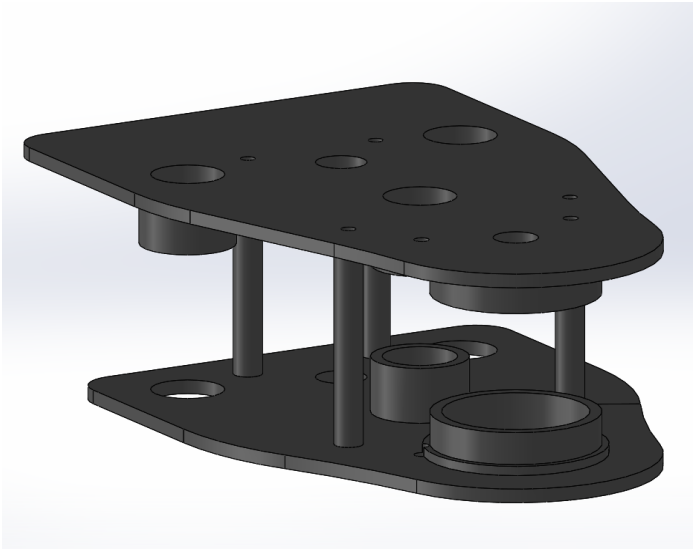
Vrchní díl má otvory pro upevnění k lineárnímu vedení, pro uchycení motoru a pro uchycení dílů, ke kterým jsou uchyceny koncové spínače. Díl je také opatřen výstupem s otvorem pro čidlo HC-SR04. Spojením vrchního dílu s lineárním vedením dosahuje základna dostatečné tuhosti.



Obrázek 6.3: Základna modelu

6.1.3.2 První rameno

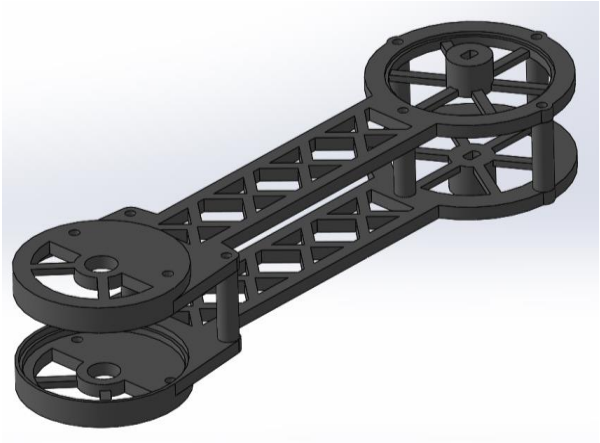
První rameno je tvořeno dvěma deskami, které jsou propojeny distančními sloupky o délce 44 mm pomocí šroubů. Každá deska má výstupy s otvory pro uložení lineárních ložisek, které jsou zajištěny pojistnými kroužky. Dále mají desky otvor pro uložení přírubové trapézové matice a otvory pro její uchycení šroubovými spoji. Matice spolu s pohybovým šroubem zajišťují pohyb prvního ramena. Desky mají v přední části kruhové výstupy sloužící jako čepy pro nasazení ložisek 6807-2RS a vrchní deska má dva otvory pro uchycení motoru 28BYJ-48, který zajišťuje pohyb druhého ramena.



Obrázek 6.4: První rameno

6.1.3.3 Druhé rameno

Druhé rameno je tvořeno dvěma podlouhlými totožnými díly, které jsou propojeny distančními sloupky o délce 22 mm pomocí šroubů. Na jednom konci dílů jsou kruhová vybrání sloužící pro uložení ložisek nasazených na prvním ramenu a ve středu vybrání je díra pro nasazení na hřídel motoru 28BYJ-48 uchyceného na prvním rameni. Na opačném konci je také kruhové vybrání, které tvoří uložení pro ložiska 6807-2RS, do kterých je nasazeno třetí rameno. Díly jsou opatřeny dvěma otvory pro montáž motoru 28BYJ-48, který zajišťuje pohyb třetího ramena.

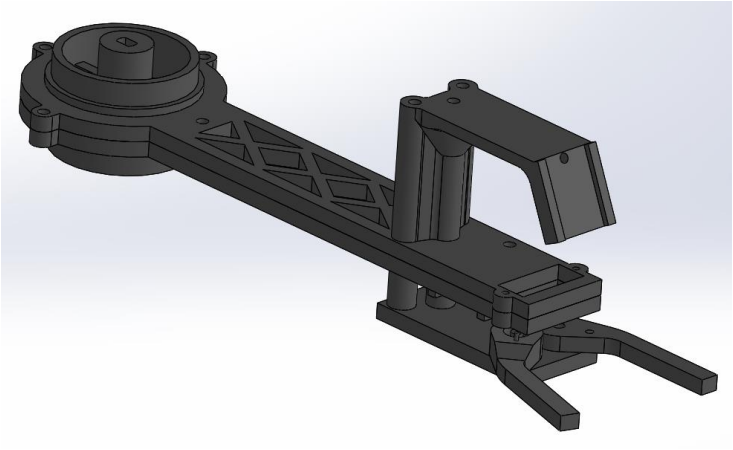


Obrázek 6.5: Druhé rameno

6.1.3.4 Třetí rameno

Třetí rameno je tvořeno dvěma podlouhlými díly, které jsou šroubově spojeny. Na jedné straně dílů jsou kruhové výstupy sloužící jako čepy pro nasazení ložisek uložených v druhém ramenu a v jejich středu je vytvořena díra pro nasazení na hřídel motoru 28BYJ-48 uchyceného na druhém ramenu. Opačné konce dílů jsou opatřeny otvory pro montáž efektoru a držáku jednoho z infračervených čidel. Pro druhé totožné čidlo je otvor přímo v dílech třetího ramena a díly mají také otvory pro uchycení servomotoru GH-S43A, který je aktuátorem pro efektor.

Koncový efektor je tvořen dvěma rameny, které slouží k uchopení předmětu a nosným dílem, který je šrouby spojen se třetím ramenem. Jedno z ramen je spojeno s tímto dílem pomocí šroubu a matice, je tedy pevné. Druhé rameno je pohyblivé a je v něm uloženo jehlové ložisko, které je nasazené na čepu nosného dílu. K pohyblivému ramenu je přilepeno výstupní rameno servomotoru, který ramenem otáčí a tím svírá, nebo pouští předmět.



Obrázek 6.6: Třetí rameno s efektorem a držákem čidla pro kontrolu uchopení

6.1.4 Tisk, příprava a montáž

Díly byly vytištěny na tiskárně Prusa 2.5 vedoucím této práce. Během tisku nenastal žádný problém a po vytištění byly díly očištěny od vláken materiálu, které vznikají přejížděním extruderu.

6.1.4.1 Příprava

Prvním krokem bylo zkrácení vodících tyčí na požadovanou délku a vyřezání závitů na jejich konce závitovým očkem. Dalším krokem byla příprava dřevěné desky, která tvoří základní část báze. Na desce byly vyfrézovány díry pro matice s podložkami a vyvrtány otvory pro uložení tyčí a uchycení upínacího ložiska pohybového šroubu. K rozměření míst pro otvory byl použit zkonstruovaný plastový díl. Seznam veškerých dílů použitých pro stavbu modelu je přiložen v příloze A.

6.1.4.2 Montáž

Nejdříve byl k dřevěné desce uchycen spodní díl základny spolu s lineárním vedením maticemi a bylo přichyceno upínací ložisko pohybového šroubu. Dalším krokem bylo uchycení lineárních ložisek do otvorů v dílech prvního ramena nasazením pojistných kroužků. Následovalo sešroubování třetího ramena a nasazení ložisek. Poté bylo sešroubováno druhé rameno, do kterého byly na jedné straně uloženy ložiska se třetím ramenem. Dále byly na druhém konci uloženy ložiska pro spojení s prvním ramenem. Nakonec byly tyto ložiska nasazeny na díly prvního ramena a ty byly spojeny sešroubováním distančních sloupků. Tím vznikl řetězec tří ramen, který bylo potřeba nasadit na lineární vedení.

Před nasazením byla k hornímu dílu prvního ramena přišroubována trapézová matice. K dolnímu dílu byla druhá matice přišroubována v takovém natočení, aby stoupání matic nebyly pootočeny a šroub se při otáčení mohl volně pohybovat. Poté byl komplet nasazen na lineární vedení a došlo k uchycení pohybového šroubu v upínacím ložisku. Následně byla k pohybovému šroubu upnuta pružná spojka a na horní závity vedení nasazeny matice a podložky pro uchycení vrchního dílu základny.

K vrchnímu dílu základny byl uchycen motor SX17-1005VLQCEF a vrchní díl včetně motoru byl nasazen na lineární vedení a přichycen k němu v požadované vzdálenosti od dřevěné desky. Poté byla k hřídeli motoru upnuta druhá strana pružné spojky. Postupně byly ke konstrukci přišroubovány jednotlivé aktuátory, koncový efektor, držáky pro jednotlivá čidla a v poslední řadě samotná čidla. Dále byl vazelínou namazán trapézový šroub a lineární vedení. Tím byla dokončena montáž zkonstruované robotické ruky.

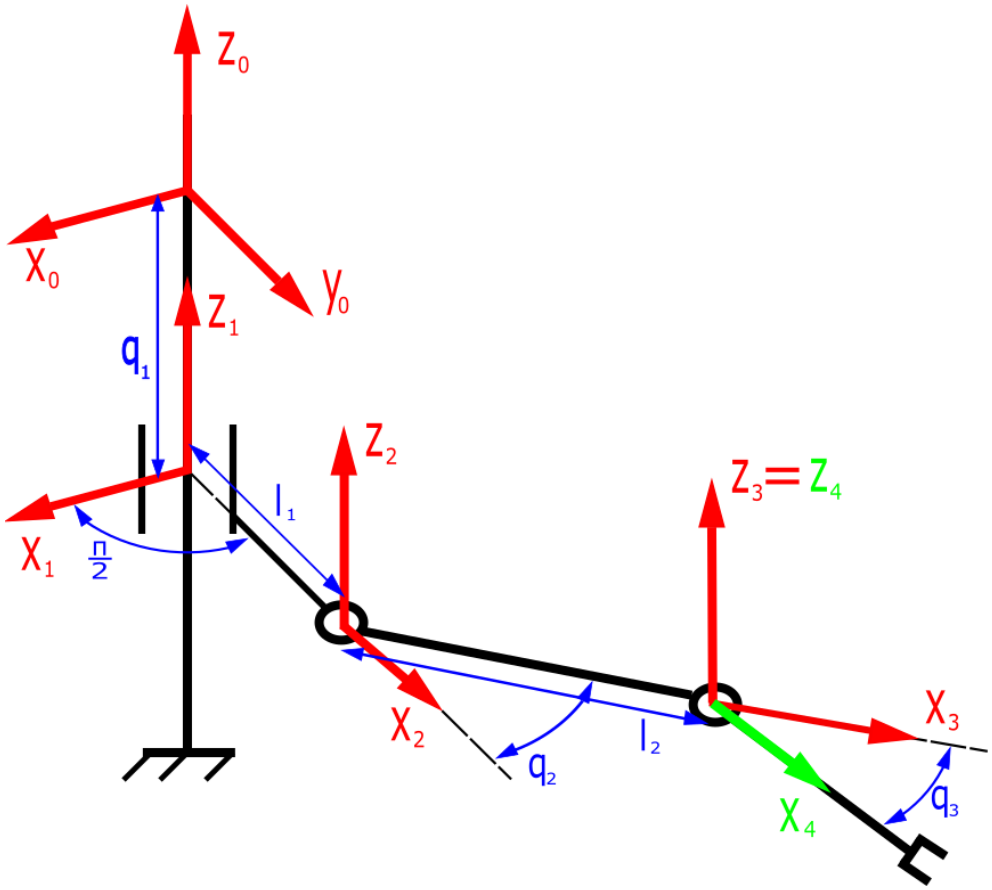
6.1.5 Kinematika modelu

V tomto úseku budou ukázány výpočty přímé a inverzní kinematiky zkonstruovaného modelu. K výpočtu přímé kinematiky bylo použito Denavit-Hartenbergových parametrů, inverzní kinematika byla vypočítána analyticky použitím geometrických vztahů mezi jednotlivými rameny.

6.1.5.1 Přímá kinematika

Při řešení přímé kinematiky sestrojené robotické ruky byly nejprve rozmístěny souřadné systémy podle D-H konvence a dále byly podle postupu popsaném v teoretické části určeny jednotlivé parametry. Rozmístění systémů je ukázáno na obrázku č. 6.7. Na obrázku jsou zobrazeny i posuny, pootočení a známé kloubové souřadnice transformací mezi jednotlivými souřadnými systémy. Denavit-Hartenbergovi parametry modelu robotické ruky pro znázorněné rozmístění souřadných systémů jsou zapsány v tabulce č. 6.1.

Souřadný systém posledního ramena ($x_4y_4z_4$) je namísto spojení s efektorem záměrně umístěn do kloubu, jelikož efektor není ve výpočtech vždy považován za koncový bod robotu. Při zjišťování obsazení pozice je jako koncový bod bráno čidlo FC-51 a ve výpočtech jsou do vztahů 14 a 15 dosazovány potřebné souřadnice x_4 a y_4.



Obrázek 6.7: Rozmístění souřadných systémů podle D-H konvence

i	ϑ_i	d_i	a_i	α_i
0	0	l_0	0	0
1	$\pi/2$	0	l_1	0
2	q_2	0	l_2	0
3	q_3	0	0	0

Tabulka 6.1: Určené Denavit-Hartenbergovi parametry modelu

Dalším krokem je sestavení transformačních matic ze systému 0 do systému 4. Transformace z jednoho systému do systému sousedního je popsána čtyřmi maticemi.

$$T_{\vartheta_i} = \begin{bmatrix} \cos\vartheta_i & -\sin\vartheta_i & 0 & 0 \\ \sin\vartheta_i & \cos\vartheta_i & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1}$$

$$T_{d_i} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2}$$

$$T_{a_i} = \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3}$$

$$T_{\alpha_i} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha_i & -\sin\alpha_i & 0 \\ 0 & \sin\alpha_i & \cos\alpha_i & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4}$$

Výsledná transformační matice mezi sousedními souřadnými systémy vznikne vynásobením jednotlivých matic v pořadí jejich pohybů.

$$T_{i-1,i} = T_{\vartheta_i} \cdot T_{d_i} \cdot T_{a_i} \cdot T_{\alpha_i} \tag{5}$$

$$T_{i-1,i} = \begin{bmatrix} \cos\vartheta_i & -\sin\vartheta_i \cdot \cos\alpha_i & \sin\vartheta_i \cdot \sin\alpha_i & a_i \cdot \cos\vartheta_i \\ \sin\vartheta_i & \cos\vartheta_i \cdot \cos\alpha_i & -\cos\vartheta_i \sin\alpha_i & a_i \cdot \sin\vartheta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6}$$

Celková transformační matice ze systému 0 do systému 4 vznikne vynásobením transformačních matic ze sousedních systémů v pořadí, v jakém byly tyto transformace prováděny.

$$T_{0,4} = T_{0,1} \cdot T_{1,2} \cdot T_{2,3} \cdot T_{3,4} \tag{7}$$

Přepočet souřadnic ze systému 4 do základního souřadného systému 0 je dán vztahem:

$$\begin{bmatrix} x_0 \\ y_0 \\ z_0 \\ 1 \end{bmatrix} = T_{0,4} \cdot \begin{bmatrix} x_4 \\ y_4 \\ z_4 \\ 1 \end{bmatrix} \tag{8}$$

Ze zjištěných Denavit-Hartenbergových parametrů byly sestaveny čtyři transformační matice pro transformace mezi souřadnými systémy. Jelikož $\cos 0 = 1$, $\sin 0 = 0$, $\sin\frac{\pi}{2} = 1$ a $\cos\frac{\pi}{2} = 0$, byly matice dále zjednodušeny pro usnadnění následného násobení.

$$T_{0,1} = \begin{bmatrix} \cos 0 & -\sin 0 \cdot \cos 0 & \sin 0 \cdot \sin 0 & 0 \cdot \cos 0 \\ \sin 0 & \cos 0 \cdot \cos 0 & -\cos 0 \sin 0 & 0 \cdot \sin 0 \\ 0 & \sin 0 & \cos 0 & q_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & q_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{9}$$

$$T_{1,2} = \begin{bmatrix} \cos\frac{\pi}{2} & -\sin\frac{\pi}{2} \cdot \cos 0 & \sin\frac{\pi}{2} \cdot \sin 0 & l_1 \cdot \cos\frac{\pi}{2} \\ \sin\frac{\pi}{2} & \cos\frac{\pi}{2} \cdot \cos 0 & -\cos\frac{\pi}{2} \cdot \sin 0 & l_1 \cdot \sin\frac{\pi}{2} \\ 0 & \sin 0 & \cos 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & l_1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{10}$$

$$T_{2,3} = \begin{bmatrix} \cos q_2 & -\sin q_2 \cdot \cos 0 & \sin q_2 \cdot \sin 0 & l_2 \cdot \cos q_2 \\ \sin q_2 & \cos q_2 \cdot \cos 0 & -\cos q_2 \sin 0 & l_2 \cdot \sin q_2 \\ 0 & \sin 0 & \cos 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} =$$

$$= \begin{bmatrix} \cos q_2 & -\sin q_2 & 0 & l_2 \cdot \cos q_2 \\ \sin q_2 & \cos q_2 & 0 & l_2 \cdot \sin q_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{11}$$

$$T_{3,4} = \begin{bmatrix} \cos q_3 & -\sin q_3 \cdot \cos 0 & \sin q_3 \cdot \sin 0 & 0 \cdot \cos q_3 \\ \sin q_3 & \cos q_3 \cdot \cos 0 & -\cos q_3 \sin 0 & 0 \cdot \sin q_3 \\ 0 & \sin 0 & \cos 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} =$$

$$= \begin{bmatrix} \cos q_3 & -\sin q_3 & 0 & 0 \\ \sin q_3 & \cos q_3 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{12}$$

Dále bylo provedeno maticové násobení a zjednodušení pomocí trigonometrických vzorců. Tím byla získána celková transformační matice.

$$T_{0,4} = \begin{bmatrix} -\sin(q_2 + q_3) & -\cos(q_2 + q_3) & 0 & -l_2 \cdot \sin q_2 \\ \cos(q_2 + q_3) & -\sin(q_2 + q_3) & 0 & l_1 + l_2 \cdot \cos q_2 \\ 0 & 0 & 1 & q_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{13}$$

Po dosazení matice $T_{0,4}$ do vztahu 8 byly vyjádřeny výsledné vzorce pro výpočet jednotlivých souřadnic a tím byla vyřešena přímá kinematika modelu.

$$x_0 = (-x_4 \cdot \sin(q_2 + q_3)) - y_4 \cdot \cos(q_2 + q_3) - l_2 \cdot \sin q_2 \tag{14}$$

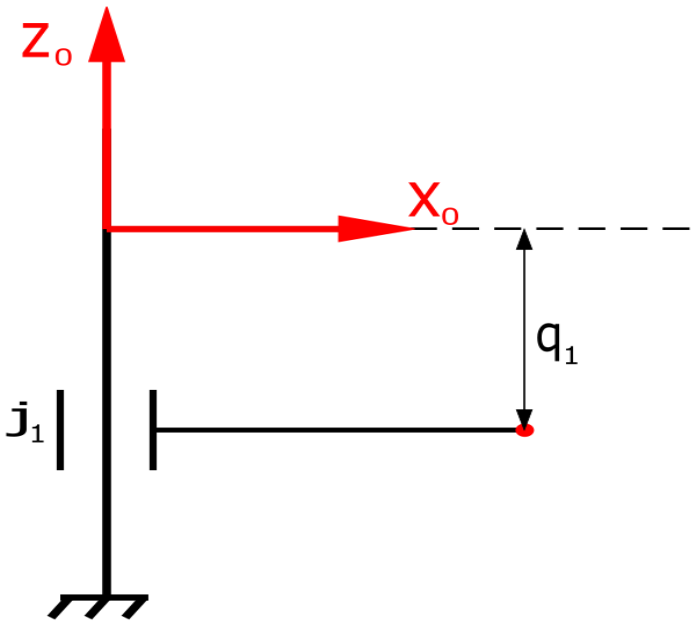
$$y_0 = x_4 \cdot \cos(q_2 + q_3) - y_4 \cdot \sin(q_2 + q_3) + l_1 + l_2 \cdot \cos q_2 \tag{15}$$

$$z_0 = z_4 + q_1 \tag{16}$$

6.1.5.2 Inverzní kinematika

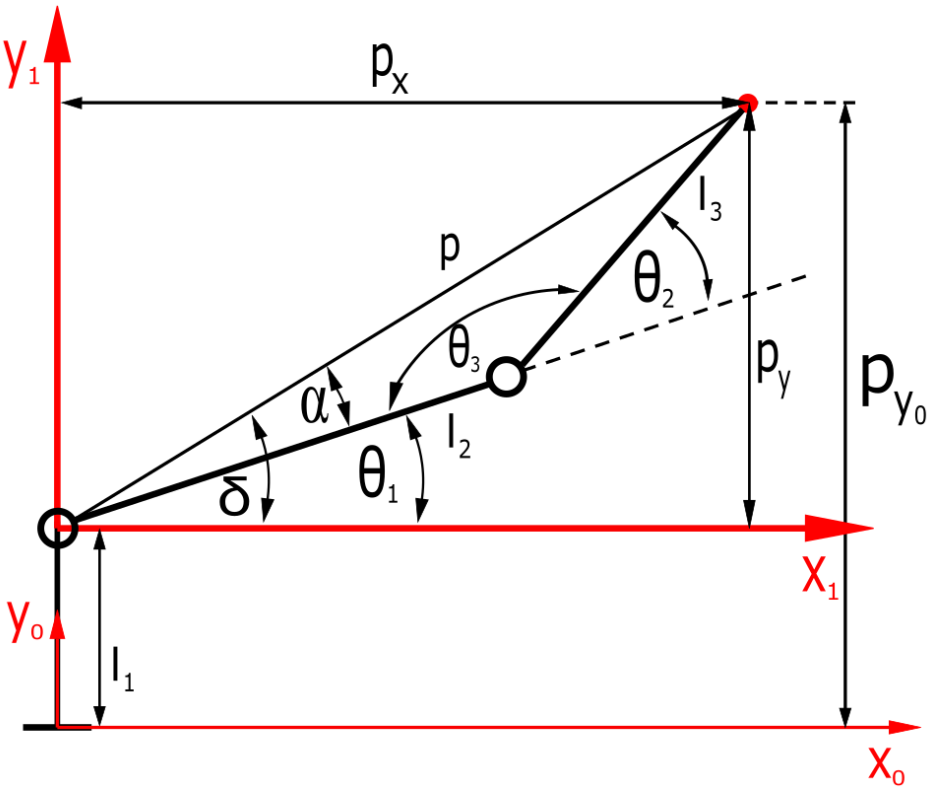
Díky jednoduché kinematické struktuře sestrojeného modelu byla inverzní kinematika řešena analyticky z geometrických vztahů mezi jednotlivými členy. Nejprve byl vyřešen hledaný posuv kloubu j_1 ve směru osy z_0, reprezentovaný hledanou souřadnicí q_1. Jak plyne z obrázku č. 6.8, vzorec pro výpočet souřadnice q_1 je velice jednoduchý.

$$q_1 = z_0 \tag{17}$$



Obrázek 6.8: Výpočet souřadnice z_0

Dále bylo potřeba načrtnout schéma pro výpočet zbylých kloubových souřadnic ze známé polohy koncového bodu. Schéma je na obrázku č. 6.9. Koncový bod je reprezentován červeným kruhem, l_1 a l_2 a l_3 jsou délky ramen, úhel θ_1 reprezentuje kloubovou souřadnici q_2 a úhel θ_2 reprezentuje kloubovou souřadnici q_3. Velikost l_3 může být vzdálenost k efektoru nebo čidlu kontrolujícímu obsazenost pozice.



Obrázek 6.9: Výpočtové schéma

Poloha koncového bodu byla pro zjednodušení výpočtů řešena v souřadném systému x_1y_1, posunutém vůči systému x_0y_0 o hodnotu prvního ramena ve směry y_0. Proto bylo potřeba přepočíst ypsilonovou vzdálenost koncového bodu.

$$p_y = p_{y_0} - l_1 \tag{18}$$

Dále byla z Pythagorovy věty vyjádřena vzdálenost p.

$$p = \sqrt{{p_x}^2 + {p_y}^2} \tag{19}$$

Poté byl z kosinové věty vyjádřen úhel θ_3.

$$p^2 = {l_2}^2 + {l_3}^2 - 2l_2l_3\cos\theta_3 \tag{20}$$

$$\theta_3 = \cos^{-1}\left(\frac{{l_2}^2+{l_3}^2-p^2}{2l_2l_3}\right) \tag{21}$$

Následně byl vyjádřen úhel θ_2.

$$\theta_2 = 180° - \theta_3 \tag{22}$$

V navazujícím kroku bylo potřeba vyjádřit úhel δ.

$$\delta = \tan^{-1}\left(\frac{p_y}{p_x}\right) \tag{23}$$

V tuto chvíli se vyskytl problém v oboru hodnot funkce $\tan^{-1}$ který je $(-90°; 90°)$ a vzorec 23 je tedy možné použít jen pro první a čtvrtý kvadrant souřadného systému. Problém byl vyřešen samostatnými vzorci pro druhý (24) a třetí (25) kvadrant.

$$\delta = 180° - \tan^{-1}\left(\frac{|p_y|}{|p_x|}\right) \tag{24}$$

$$\delta = 180° + \tan^{-1}\left(\frac{|p_y|}{|p_x|}\right) \tag{25}$$

Dále byl pomocí sinové věty a známé vzdálenosti p a úhlu θ_3 vyjádřen úhel α.

$$\frac{l_3}{\sin\alpha} = \frac{p}{\sin\theta_3} \tag{26}$$

$$\alpha = \sin\left(\frac{l_3}{p}\cdot\sin\theta_3\right) \tag{27}$$

Posledním krokem bylo vyjádřit úhel θ_1.

$$\theta_1 = \delta - \alpha \tag{28}$$

Tím byly vyjádřeny potřebné vzorce pro výpočet všech kloubových souřadnic. Bohužel, tento postup zahrnuje jen výpočty pro první, druhý a třetí kvadrant. Úhel θ_2 je v předešlých případech počítán jako kladný, aby bylo vždy jen jedno řešení nastavení ramen pro zadané hodnoty.

Z konstrukce modelu vzniká omezení pohybu ramen. Druhé rameno je omezeno pohybem v rozmezí –90° až 90° od osy x_2 z obrázku č. 6.6, třetí rameno se může pohybovat v rozmezí přibližně –120° až 120° od osy x_3 ze stejného obrázku. Z omezení těchto pohybů vyplívá, že se pro manipulaci ve čtvrtém kvadrantu a oblasti $(x_1 - 149)^2 + {y_1}^2 \leq l_3$ (značení os z obrázku č. 6.8) bude muset třetí rameno otáčet v záporném směru.

Výpočty úhlu θ_3, úhlu α, úhlu δ a úhlu θ_2 jsou pro tuto situaci totožné jako v předchozím případě až na rozdíl v úhlu θ_2, který je zde otočen a zvětšuje v záporném směru. Pro výpočty v řídícím programu bude úhel ošetřen znaménkem minus. Rozdíl je ve výpočtech úhlu θ_1 pro čtvrtý kvadrant (29) a výše zmíněnou oblast (30).

$$\theta_1 = \alpha - \delta \tag{29}$$

$$\theta_1 = \alpha + \delta \tag{30}$$

Tím byla vyřešena inverzní kinematika sestrojeného modelu robotické ruky.

6.1.5.3 Pracovní prostor

Pracovní prostor sestrojeného modelu je při pohledu z kladné části osy z_0 znázorněn na obrázku č. 6.10. Prostor je při tomto pohledu necelým mezikružím, omezeným konstrukčními podmínkami robotu. Vnitřní okraj mezikruží je omezen krajní hodnotou natočení třetího ramena. Vnější okraj není omezen dosahem efektoru, ale dosahem čidla, které kontroluje, zda se na zadaných souřadnicích nachází předmět. Čidlo i efektor jsou umístěny na třetím rameni, ale jelikož je čidlo ke kloubu j_2 umístěno blíž, vyplívá omezení ze vzdálenosti čidla od tohoto kloubu ($l_{3č}$).

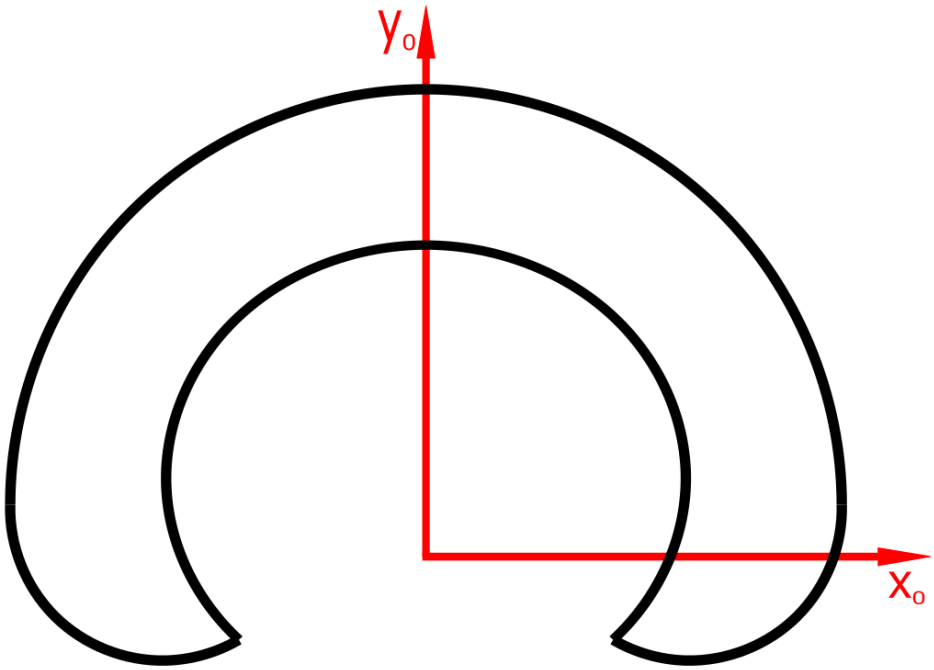
Omezení pracovního prostoru jsou definovány rovnicemi 31-34.

$${x_0}^2 + (y_0 - l_1)^2 \leq (l_2 + l_{3č})^2 \tag{31}$$

$${x_0}^2 + (y_0 - l_1)^2 \geq 145{,}77^2 \tag{32}$$

$$(x_0 - l_2)^2 + (y_0 - l_1)^2 \leq l_{3č} \tag{33}$$

$$(x_0 + l_2)^2 + (y_0 - l_1)^2 \leq l_{3č} \tag{34}$$



Obrázek 6.10: Pracovní prostor v souřadnicích x_0 a y_0

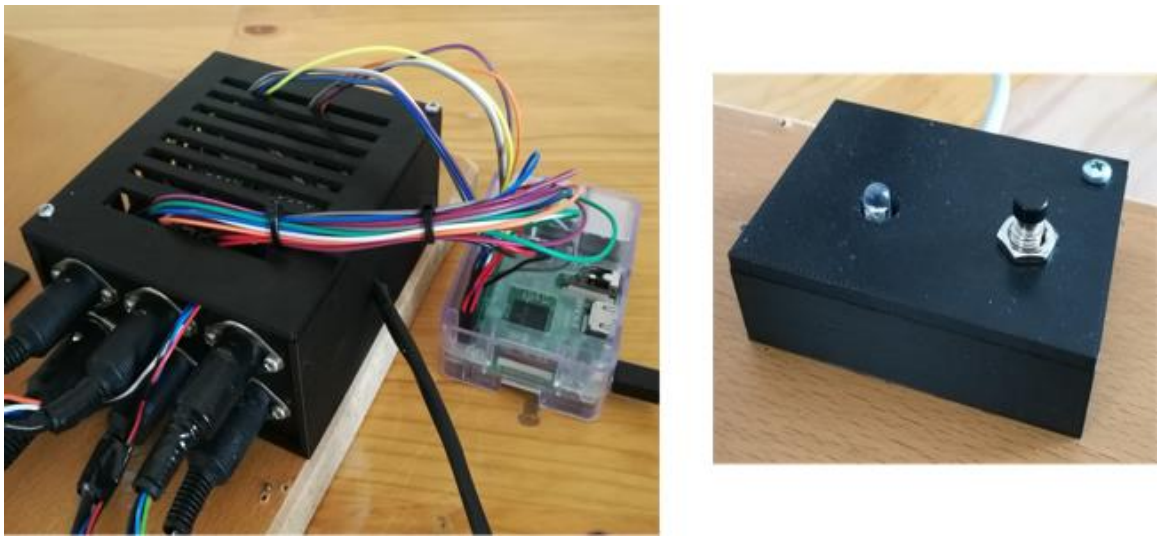
Pracovním prostorem sestrojeného modelu je tedy obrazec z obrázku č. 6.10, který je ve směru osy z_0 protažen ve 3D prostor. V tomto směru je pracovní prostor omezen maximálním a minimální hodnotou posunu prvního kloubu.

6.2 Zapojení hardwaru

Dalším krokem praktické části bylo zapojení hardwaru. Součástky byly zapojovány dle značení jejich výstupních pinů a návodů.

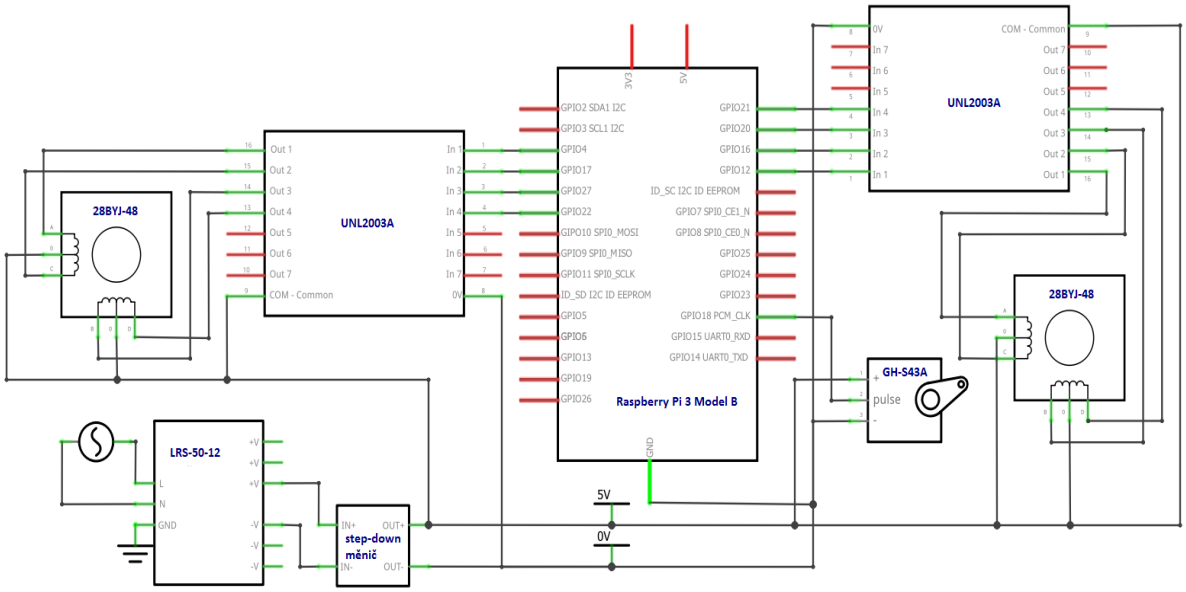
Pro uschování řídící elektroniky (ovladače motorů, měniče napětí atd.) byla zkonstruována krabička (na obrázku č. 6.11 vlevo), ze které jsou vyvedeny DIN konektory pro připojení vstupních a výstupních periferií robotu, což poskytuje částečnou modulárnost zapojení. Také je v krabičce pět rozbočovacích můstků, na které jsou přivedeny napětí a zem z Raspberry Pi a napětí a zem ze snižovacího měniče napětí, protože je k nim potřeba připojit větší množství kabelů. Dále byla zkonstruována druhá krabička, která je menší a slouží pro komunikaci s obsluhou robotu (LED modul, tlačítko)

Po vytištění krabiček byly přilepeny rozbočovací můstky na dno větší krabičky, napájeny kabely k DIN konektorům, a to jak kabely jdoucí od komponent na robotu a komunikační krabičky, tak kabely vedoucí k Raspberry Pi a součástkám uvnitř větší krabičky. Obě krabičky pak byly spolu s víky přišroubovány ke dřevěné desce a byly propojeny jednotlivé konektory a připojeny kabely k GPIO rozhraní a zdroji.



Obrázek 6.11: Fotografie krabiček

Schéma zapojení bylo pro jeho velikost a tím snížení čitelnosti v této práci záměrně rozděleno do tří schémat. Pro nakreslení schémat byl použit program Fritzing. První ze schémat, obrázek č. 6.12, znázorňuje zapojením servomotoru a motorů 28BYJ-48 s jejich ovladači.



Obrázek 6.12: Schéma zapojení motorů 28BYJ-48 s jejich ovladači a servomotoru

Druhé schéma, obrázek č. 6.13, představuje zapojení motoru SX17-1005VLQCEF včetně jeho ovladače a koncových spínačů. Před ovladač A4988 je podle návodu umístěn elektrolytický kondenzátor o kapacitě 100 µF. Na ovladači je potřeba před připojením motoru nastavit omezení proudu tekoucího skrz vinutí motoru. To je provedeno nastavením požadovaného napětí na referenčním pinu trimrem. Napětí je vypočítané podle vzorce 35 kde

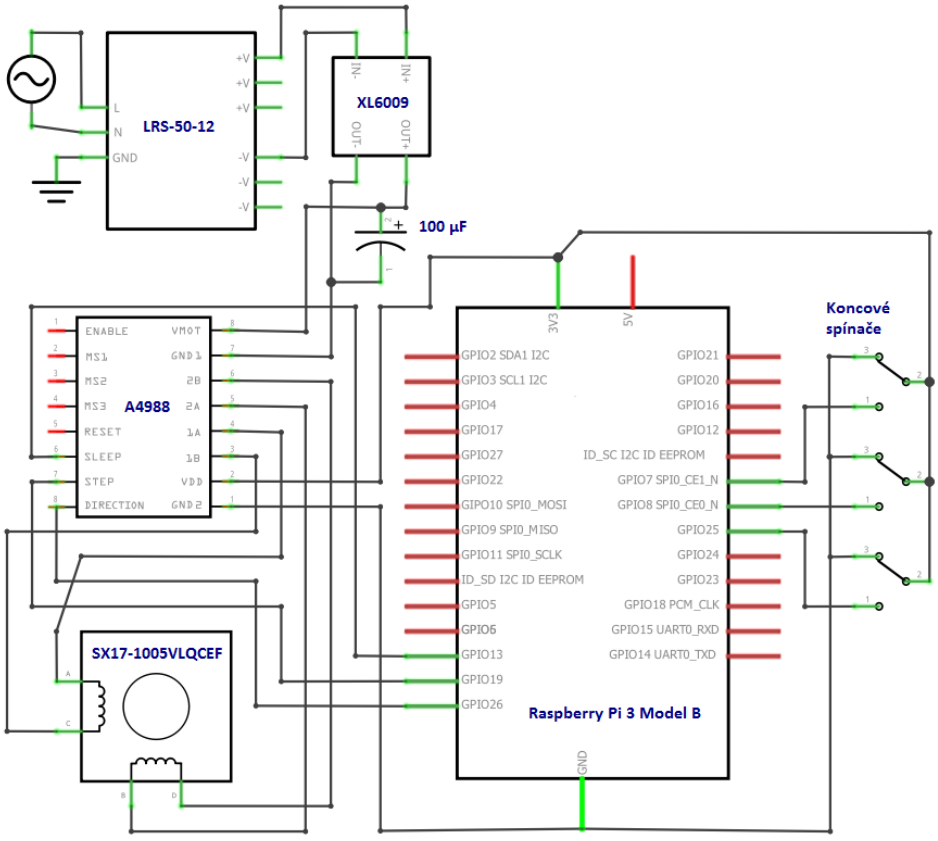
I, je požadovaná hodnota proudu tekoucího vynutími motoru (pro motor SX17-1005VLQCEF je hodnota 1 A) a R_S je hodnota odporu sloužícího ke snímání proudu (v případě použitého ovladače je hodnota 0,1 Ω).

$$V_{ref} = I \cdot 8 \cdot R_S \tag{35}$$

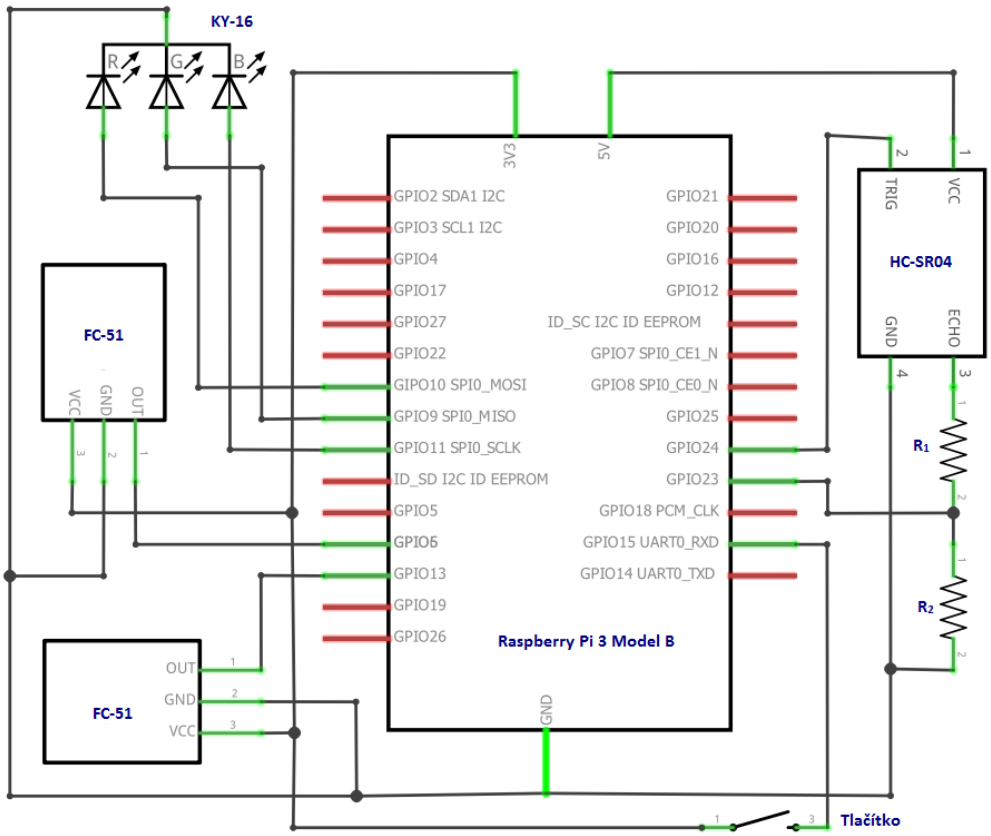
Na obrázku č. 6.14 je poslední schéma znázorňující zapojení senzorů, RGB modulu a tlačítka. Jelikož je senzor HC-SR04 napájen hodnotou 5 V, je potřeba ke snížení napětí pro pin na Raspberry Pi použít napěťový dělič. Z vyjádřeného vztahu 37 po dosazení hodnot U = 5 V a U_1 = 3,3 V vyplývá, že rezistor R_1 je dvojnásobkem hodnoty odporu R_2. Z toho důvodu byly použity rezistory o hodnotách 1 kΩ a 2 kΩ.

$$U_1 = U \cdot \frac{R_1}{R_1 + R_2} \tag{36}$$

$$R_1 = \frac{U_1}{U - U_1} \cdot R_2 \tag{37}$$



Obrázek 6.13: Schéma zapojení SX17-1005VLQCEF s ovladačem a koncovými spínači



Obrázek 6.14: Schéma zapojení senzorů, tlačítka a RGB modulu

6.3 Software

6.3.1 Instalace Ubuntu Core 16

Pro instalaci operačního systému Ubuntu Core 16 bylo potřeba k Raspberry Pi připojit obrazovku a klávesnici kvůli prvnímu spuštění a nastavení systému. Před samotnou instalací bylo nutné provést několik kroků, bez kterých by spuštění systému nemohlo proběhnout.

Nejdříve byl vytvořen účet Ubuntu One, který slouží pro přístup k softwaru Ubuntu. Po vytvoření účtu byla v programu PuTTYgen vytvořena dvojice SSH klíčů, sloužící pro připojování k operačnímu systému. Veřejný klíč byl následně nahrán na vytvořený účet. Posledním krokem před instalací bylo stažení obrazu operačního systému ze stránek Ubuntu a zkopírování obrazu pomocí programu Win32DiskImager na SD kartu zakoupenou pro Raspberry Pi.

V tuto chvíli bylo Raspberry Pi ethernetovým kabelem připojeno k internetu, byla vložena SD karta s obrazem operačního systému a do Raspberry Pi byl zasunut konektor ze zdroje. Následně začalo první spouštění systému, přičemž bylo následováno pokynů na obrazovce pro

konfiguraci systému. Bylo automaticky nakonfigurováno síťové připojení a následně byla vložena e-mailová adresa použitá při vytvoření účtu Ubuntu One. Tím se ze serveru Ubuntu nahrál do Raspberry Pi veřejný SSH klíč uložený na účtu, který slouží pro přihlašování k operačnímu systému. Následně byla konfigurace dokončena a na obrazovce se zobrazilo uživatelské jméno s IP adresou pro vzdálené připojení k operačnímu systému, čímž byla dokončena jeho instalace.

Pro přihlašování do systému pomocí privátního SSH klíče je používán program PuTTY, do kterého jsou vloženy přihlašovací údaje zobrazené na obrazovce a cesta k privátnímu klíči. Při přihlašovaní je potřeba zadat heslo k privátnímu klíči a následně dojde ke spuštění operačního systému.

6.3.2 Instalace ROS a potřebných balíků

Ubuntu Core má možnost instalovat pouze snap balíčky. Proto byl nainstalován balík classic, který vytvoří klasické prostředí Ubuntu, které nabízí správu balíků pomocí klasických nástrojů jako např. apt a poskytuje přístup k veškerému hardwaru zařízení.

```
$ snap install classic --edge --devmode
```

Prostředí je potřeba rozbalit při každém zapnutí operačního systému a spuštění nového okna.

```
$ sudo classic
```

Poté už je možno použít příkaz apt při instalaci modulu RPi.GPIO.

```
$ sudo apt install python-dev python-pip python-setuptools
$ pip install RPi.GPIO
```

6.3.2.1 Instalace ROS

Systém je třeba nastavit pro přijímaní balíčků ROS a je důležité přidat klíč který ověřuje, zda přijímané balíčky v úložišti skutečně pocházejí z oficiálních zdrojů ROS. Poté je nutné provést aktualizaci indexování balíčků.

```
$ sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu
$(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'
$ sudo apt-key adv --keyserver hkp://ha.pool.sks-keyservers.net:80
--recv-key 421C365BD9FF1F717815A3895523BAEEB01FA116
$ sudo apt-get update
```

Následuje instalace ROS Kinetik Kame a inicializace rosdep, který je vyžadován pro správnou funkci systému.

```
$ sudo apt-get install ros-kinetic-ros-base
$ sudo rosdep init
$ rosdep update
```

V tuto chvíli je ROS nainstalován a dále je změněn soubor .bashrc v domovském adresáři, čímž je zajištěna aktivace pracovního prostoru při každém startu systému nebo spuštění nového okna.

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

Následně je ještě potřeba doinstalovat několik nástrojů, sloužících pro tvorbu vlastních balíků a pracovního prostředí ROS.

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

Nyní je na Raspberry Pi nainstalován ROS i se všemi potřebnými balíky a moduly pro následné vytváření uzlů a ovládání rozhraní GPIO.

6.3.3 Pracovní prostředí a balík pro projekt

Po nainstalování ROS je v domovském adresáři vytvořen adresář bakalarka, ve kterém je vytvořeno pracovní prostředí ROS, nazvané catkin_ws, uvnitř kterého je vytvořen adresář src.

```
$ mkdir ~/bakalarka
$ mkdir -p ~/bakalarka /catkin_ws/src
$ cd ~/bakalarka /catkin_ws/
$ catkin_make
```

Dále je zapotřebí přesunout se do adresáře src, což je adresář obsahující zdrojové kódy balíků a v něm vytvořit balík pro řízení robotické ruky. V tomto adresáři je potřeba použít příkaz pro vytvoření souboru CMakeLists.txt, který je důležitý pro konfiguraci projektů. Následně je vytvořen balík pro model robotické ruky. Za příkazem pro vytvoření je jeho jméno, které následují závislosti pro psaní zdrojových kódů. Uvnitř balíku r_ruka je poté vytvořen adresář skripty, ve kterém jsou uloženy programy jednotlivých uzlů.

```
$ cd src
$ catkin_init_workspace
$ catkin_create_pkg r_ruka rospy std_msgs python-rpi.gpio
```

```
$ cd r_ruka
$ mkdir skripty
```

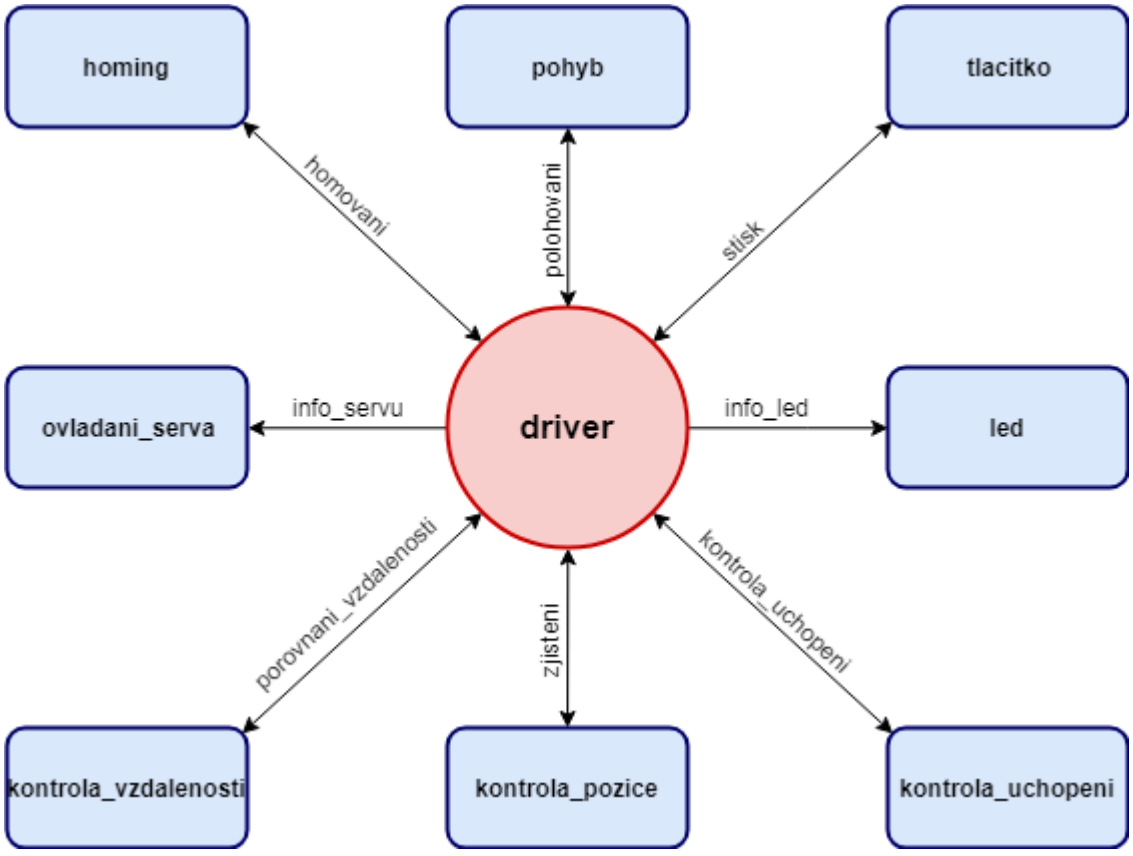
V tuto chvíli je vytvořené vše potřebné pro napsaní a spuštění uzlů ROS.

6.3.4 Řízení modelu robotické ruky

Jak bylo zmíněno v teoretické části, ROS funguje jako soustava uzlů, které mezi sebou vzájemně komunikují. Řízení modelu je tedy rozděleno do devíti uzlů, z nichž jeden je uzlem řídícím celé chování robotu a ostatní slouží pro ovládání jednotlivých výstupů, případně načítaní hodnot ze vstupů. Na obrázku č. 6.13 je zobrazené schéma komunikace mezi uzly.

Na začátku každého uzlu jsou importovány potřebné moduly a soubory pro komunikaci v systému ROS. Příkladem je import z uzlu driver.

```
import rospy
import time
import math
from sys import exit
from std_msgs.msg import String
from r_ruka.srv import *
```



Obrázek 6.1: Komunikace mezi uzly

Následuje deklarace proměnných představujících použité piny rozhraní GPIO v tomto uzlu, případně deklarace globálních proměnných použitých v daném uzlu. Následuje první funkce, která slouží pro nastavení rozhraní GPIO. Příklad je z uzlu pohyb.

```
direction = 26
step = 19
sleep = 13
endstop_nema = 7
endstop1 = 25
endstop2 = 8
krokac1 = [4, 17, 27, 22]
krokac2 = [12, 16, 20, 21]
delay_nema = 0.00101
delay_krokace = 0.002

def nastav_GPIO():
    GPIO.setmode(GPIO.BCM)
    GPIO.setwarnings(False)
    GPIO.setup(direction, GPIO.OUT)
    GPIO.output(direction, 1)
    GPIO.setup(step, GPIO.OUT)
    GPIO.output(step, 0)
    GPIO.setup(sleep, GPIO.OUT)
    GPIO.output(sleep, 0)
    GPIO.setup(endstop_nema, GPIO.IN)
    GPIO.setup(endstop1, GPIO.IN)
    GPIO.setup(endstop2, GPIO.IN)
    for pin in range(4):
          GPIO.setup(krokac1[pin],GPIO.OUT)
          GPIO.output(krokac1[pin], 0)
          GPIO.setup(krokac2[pin],GPIO.OUT)
          GPIO.output(krokac2[pin], 0)
```

6.3.4.1 driver

Řídící uzel byl pojmenován driver a funguje jako publisher, případně client pro jednotlivé uzly typu subscriber nebo service, ovládající periferie robotu. Driver je jediným uzlem, který nemá přístup k rozhraní GPIO.

Tento uzel řídí celé chování robotu. Obsahuje funkce sloužící pro výpočty počtu kroků jednotlivých motorů pro přejetí z jedné polohy do druhé a funkce počítající ze zadaných souřadnic polohy čidla souřadnice polohy předmětu pro efektor. Jako příklad je zde uvedena funkce theta1, která provádí výpočet úhlu θ_1, představující výpočet natočení druhého ramena pro zadané souřadnice. Parametry funkce jsou souřadnice a délka ramena l_3. Ve funkci jsou

podmínkami ošetřeny výpočty pro jednotlivé kvadranty a oblast zmíněnou ve výpočtu inverzní kinematiky.

```
def theta1(x, y, rameno_2):
    theta_3 = theta3(prepona(x, y), rameno_2)
    uhel_xy = float
    if ((x - 149)**2 + y**2 <= rameno_2**2):
          uhel_xy = math.degrees(math.atan(y/x))
          theta_1 = ( uhel_xy + math.degrees(math.asin
(rameno_2/prepona(x, y)*math.sin(math.radians(theta_3)))) )
    elif ( (x > 0) and (y < 0) ):
          y = -y
          uhel_xy = math.degrees(math.atan(y/x))
          theta_1 = ( math.degrees(math.asin ( rameno_2/prepona(x,
y)*math.sin( math.radians(theta_3)))) - uhel_xy )
    else:
          if (x >= 0 and y >=0):
                uhel_xy = math.degrees(math.atan(y/x))
          elif (x <= 0 and y >=0):
                x = -x
                uhel_xy = (180 - math.degrees(math.atan(y/x)))
          else:
                x = -x
                y = -y
                uhel_xy = (180 + math.degrees(math.atan(y/x)))
          theta_1 = ( uhel_xy - math.degrees(math.asin
(rameno_2/prepona(x, y)*math.sin(math.radians(theta_3)))) )
    return theta_1 - 90
```

V hlavní části programu je cyklus celého řízení, zobrazený na obrázku č. 6.14. Při chodu programu dochází ke komunikaci s ostatními uzly, zajišťujícími vykonání potřebného úkonu, nebo zjištění určitého stavu např. zda se na zadané pozici nachází předmět.

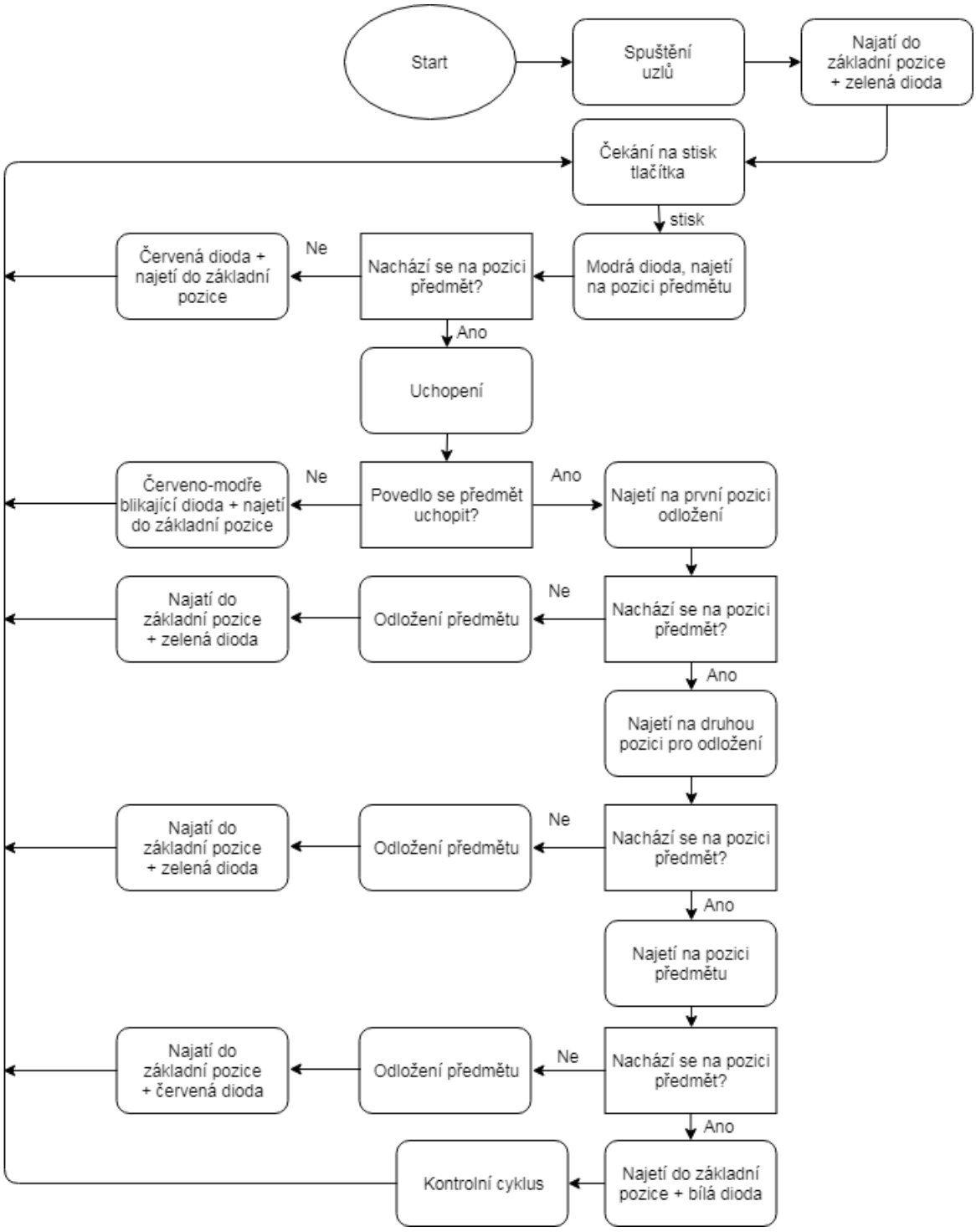
Uzel driver obsahuje inicializaci v systému ROS jakožto publishera pro uzly ovladani_serva a led. Inicializace uzlu publishera se provádí podle šablony, která obsahuje název tématu a složku pomocí které komunikuje publisher se subscriberem. Následuje pojmenování uzlu a anonymous = True zajistí jedinečnost uzlu přidáním náhodných čísel za jeho jméno.

```
servo_pub = rospy.Publisher('info_servu', String)
led_pub = rospy.Publisher('info_led', String)
rospy.init_node('driver', anonymous = True)
```

Pro odesílaní zpráv uzlům ovladani_serva a led jsou vytvořeny samostatné funkce, volané při běhu programu.

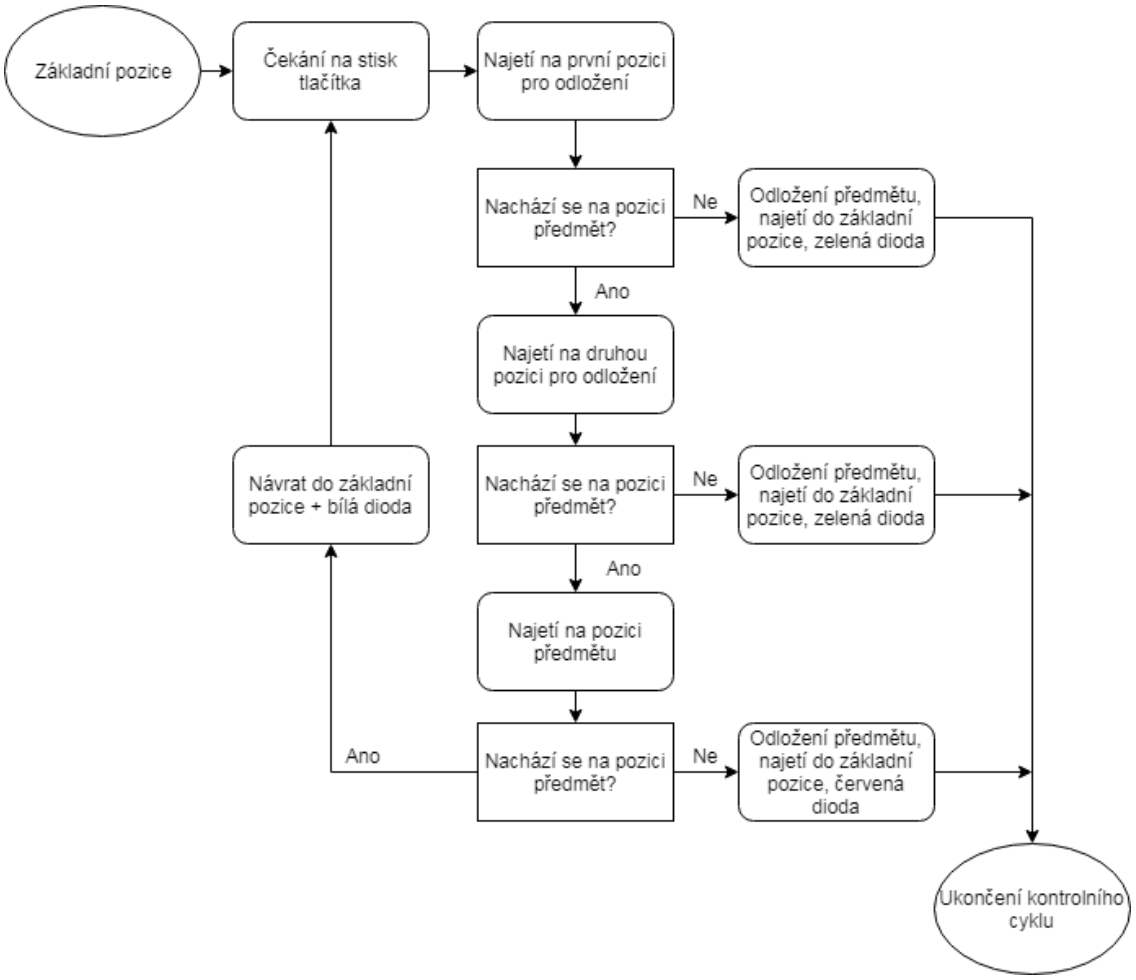
```
def publisher_servo(info_pro_servo):
    servo_pub.publish(info_pro_servo)

def publisher_ledka(info_pro_ledku):
    led_pub.publish(info_pro_ledku)
```



Obrázek 6.2: Cyklus chodu modelu

Průběh kontrolního cyklu je ukázán na obrázku č. 6.15.



Obrázek 6.3: Kontrolní cyklus

Uzel také slouží jako client k uzlům kontrola_pozice, kontrola_uchopeni, kontrola_vzdalenosti a tlacitko. Při komunikaci s těmito uzly je zapotřebí zpětná vazba, jelikož se jedná o uzly zjišťující určitý stav, který rozhoduje o dalším běhu řídícího programu. Driver je také clientem pro uzly homing a pohyb. V tomto případě se jedná o uzly řídící pohyb ramen a zpětná vazba je zde proto, aby cyklus programu nepokračoval před dokončením najíždění na požadované souřadnice.

Deklarace funkce pro volání služby je ukázána na funkci volající službu s názvem polohovani, která reprezentuje uzel pohyb. Pro uzel client není potřeba použít funkci init_node. První řádek funkce je metoda, která čeká, dokud se požadovaná služba neobjeví v systému. Třetí řádek je šablona volání služby, která je ve čtvrtém řádku použita jako funkce s parametry obsahujícími počet kroků pro jednotlivé motory. Šablona má dva parametry, a to název služby a název souboru pro předávání zpráv. Čtvrtý řádek vrací výsledek zaslaný službou polohovaní.

```
def najeti_client(pocty_kroku):
    rospy.wait_for_service('polohovani')
    try:
          najed = rospy.ServiceProxy('polohovani', polohovani)
          resp = najed(pocty_kroku[0], pocty_kroku[1],
pocty_kroku[2])
          return resp.vysledek
    except rospy.ServiceException, e:
          return 'chyba'
```

Funkce najeti_client je pak v běhu programu volána pro najetí robotu o vypočítané kroky jednotlivých motorů. Kompletní výpis zdrojového kódu uzlu driver je přiložen v příloze B.

6.3.4.2 ovladani_serva

Tento uzel řídí natočení servomotoru GH-S43A, který pohybuje ramenem efektoru, čímž se stará o uchopení předmětu. Uzel je subscriberem pro uzel driver. Při inicializaci je uzel pojmenován a následně deklarován podle šablony, která obsahuje název tématu, soubor pro komunikaci a funkci, kterou při přijetí zprávy volá. Poslední řádek deklarace uzlu subscriber obsahuje metodu, která zajistí neustálý běh uzlu.

```
def subscriber_servo():
    rospy.init_node('servo', anonymous = True)
    rospy.Subscriber('info_servu', String, ovladani)
    rospy.spin()
```

K ovládání servomotoru je použito pulzně šířkové modulace, kterou rozhraní GPIO nabízí. Podle obdržené zprávy servomotor buď předmět uchopí, nebo ho pustí. Kompletní výpis zdrojového kódu uzlu ovladani_serva je přiložen v příloze B.

```
def uchop():
    servo.ChangeDutyCycle(7.7)

def pust():
    servo.ChangeDutyCycle(6.4)

def ovladani(data):
    if data.data == 'uchop':
          uchop()
    if data.data == 'pust':
          pust()
```

6.3.4.3 led

Uzel led se stará o rozsvěcování diody na RGB modulu a změnu její barvy. V tomto případě se také jedná o uzel typu subscriber, který podle obdržené zprávy provede požadovanou změnu. Led dioda může svítit zeleně (robot připraven a čeká na pokyn), modře (robot vykonává definovaný úkon), červeně (robot na pozici nenalezl předmět, obě pozice pro odkládání obsazeny, nepodařilo se odložit předmět, protože je obsazená i původní pozice předmětu), bíle (předmět na pozici se nepodařilo uchopit) nebo bliká červeně (chyba v komunikaci, ztráta kroku motoru SX17-1005VLQCEF). Kompletní výpis zdrojového kódu uzlu led je přiložen v příloze B.

```
def barva(data):
    GPIO.output(R, 0)
    GPIO.output(G, 0)
    GPIO.output(B, 0)
    if data.data == 'zelena':
          GPIO.output(G, 1)
    elif data.data == 'modra':
          GPIO.output(B, 1)
    elif data.data == 'cervena_blik':
          while (1):
                GPIO.output(R, 1)
                time.sleep(1)
                GPIO.output(R, 0)
                time.sleep(1)

    elif data.data == 'neuchopeno':
                GPIO.output(R, 1)
                GPIO.output(G, 1)
                GPIO.output(B, 1)
    else:
          GPIO.output(R, 1)
```

6.3.4.4 pohyb

Tento uzel se stará o pohyb modelu. Pohyb je uzlem typu service, jehož deklarace je k vidění ve funkci polohovani_service. Inicializace uzlů typu service probíhá podle stanovené šablony. Druhý řádek obsahuje deklaraci služby, při které je zadán její název, soubor pro komunikaci a funkce, která je službou volána.

```
def najed_do_pozice(req):
    najeti_do_pozice(req.nema, req.krokac1, req.krokac2)
    return polohovaniResponse("Hotovo")
```

```
def polohovani_service():
    rospy.init_node('polohovani')
    s = rospy.Service('polohovani', polohovani, najed_do_pozice)
    rospy.spin()
```

Zpráva kterou obdrží v souboru polohovani jsou kroky pro motory, ze kterých vyhodnotí průběh pohybu. Znaménkem počtu kroků je určen směr otáčení, případně posuvu. Jestliže je směr posuvu prvního ramena dolů, provedou se nejdříve pohyby druhého a třetího ramena. V případě že je směr posuvu vzhůru, provede se nejdříve pohyb ramena prvního. Pohyby jsou takto vykonávány záměrně, aby robot neposouval předměty umístěné v jeho okolí. Druhé a třetí rameno se otáčejí současně do okamžiku, kdy rameno s nižším počtem kroků dosáhne požadovaného počtu kroků. Poté se pohybuje jen rameno s vyšším počtem kroků.

Pohybu motoru SX17-1005VLQCEF je ovládán nastavením směru otáčení na pinu direction a počet kroků je počet signálů odeslaných řídícímu ovladači A4988 na pin step.

```
GPIO.output(direction, 1)
time.sleep(0.005)
for i in range (kroky_nema):
    GPIO.output(step, 1)
    time.sleep(0.0000015)
    GPIO.output(step, 0)
    time.sleep(delay_nema)
```

Pohyb motorů 28BYJ-48 je zajištěn současným zasíláním vysokých a nízkých napěťových hodnot na vstupní piny ovladačů UNL2003A podle tabulky v návodu motorů. Kompletní výpis zdrojového kódu uzlu pohyb je přiložen v příloze B.

```
sekvence1 = range(0, 8)
sekvence1[0] = [1,0,0,1]
sekvence1[1] = [0,0,0,1]
sekvence1[2] = [0,0,1,1]
sekvence1[3] = [0,0,1,0]
sekvence1[4] = [0,1,1,0]
sekvence1[5] = [0,1,0,0]
sekvence1[6] = [1,1,0,0]
sekvence1[7] = [1,0,0,0]

def krok_krokac1(p1, p2, p3, p4):
    GPIO.output(krokac1[0], p1)
    GPIO.output(krokac1[1], p2)
    GPIO.output(krokac1[2], p3)
    GPIO.output(krokac1[3], p4)
```

```
if (kroky_krokac1 >= kroky_krokac2):
    for i in range(kroky_krokac2):
          for j in range(8):
                krok_krokac1(sekvence2[j][0], sekvence2[j][1],
sekvence2[j][2],sekvence2[j][3])
                krok_krokac2(sekvence1[j][0], sekvence1[j][1],
sekvence1[j][2],sekvence1[j][3])
                time.sleep(delay_krokace)
    for i in range(kroky_krokac1 - kroky_krokac2):
          for j in range(8):
                krok_krokac1(sekvence2[j][0], sekvence2[j][1],
sekvence2[j][2],sekvence2[j][3])
                time.sleep(delay_krokace)
```

6.3.4.5 homing

Uzel homing zajišťuje najetí jednotlivých ramen modelu do základní pozice. Základní pozice je jediná známá poloha modelu, takže je od ní vždy zahájen pohyb ramen. Do této pozice se také model vždy po vykonání zadaných pohybů vrací. Uzlu homing zasílá driver ve zprávě informaci, jestli najíždění pozice probíhá ze známé pozice.

Při pohybu ze známé pozice, kdy nemůže nastat kolize, je najíždění navazujícím pohybem ramen od prvního po třetí, přičemž pohybující se rameno je v základní poloze zastaveno stlačením páčky koncového spínače. V případě, že se jedná o neznámou pozici, dojde nejdříve k pootočení druhého a třetího ramena směrem od koncových spínačů, aby nedošlo ke kolizi ramen s jejich držáky. Následně proběhne najíždění do základní pozice stejně jako ze známé pozice. Kompletní výpis zdrojového kódu uzlu homing je přiložen v příloze B.

```
def homing_z_pozice():
    GPIO.output(sleep, 1)
    time.sleep(0.005)
    while (GPIO.input(endstop_nema) != 0):
          GPIO.output(step, 1)
          time.sleep(0.0000015)
          GPIO.output(step, 0)
          time.sleep(delay_nema)
    p = 8
    GPIO.output(sleep, 0)
    while (GPIO.input(endstop1) != 0):
          k = p % 8
          krok_krokac1(sekvence1[k][0], sekvence1[k][1],
sekvence1[k][2], sekvence1[k][3])
          time.sleep(delay_krokace)
          p += 1
    p = 8
```

```
    while (GPIO.input(endstop2) != 0):
        k = p % 8
        krok_krokac2(sekvence2[k][0], sekvence2[k][1],
sekvence2[k][2], sekvence2[k][3])
        time.sleep(delay_krokace)
        p += 1
```

6.3.4.6 kontrola_pozice

Kontrola_pozice je uzlem typu service, který zjišťuje, zda se na pozici nachází předmět. Načítá hodnotu z infračerveného senzoru, kterou následně vyhodnotí a výsledek odešle driveru. Kompletní výpis zdrojového kódu uzlu kontrola_pozice je přiložen v příloze B.

```
def zjisti(req):
    if (GPIO.input(infra_cidlo) == 0):
        return objektResponse('obsazeno')
    else:
        return objektResponse('prazdno')
```

6.3.4.7 kontrola_uchopeni

Kontrola_uchopení je uzel, zjišťující úspěšnost uchopení předmětu. Po obdržení požadavku ověří, zda byl předmět uchopen a odešle výsledek uzlu driver. Pro kontrolu uchopení a kontrolu předmětu na pozici jsou použity stejná čidla, takže uzly kontrola_pozice a kontrola_uchopeni jsou téměř totožné. Kompletní výpis zdrojového kódu uzlu kontrola_uchopeni je přiložen v příloze B.

6.3.4.8 kontrola_vzdalenosti

Uzel kontrola_vzdalenosti porovnává očekávanou hodnotu obdrženou od uzlu driver s hodnotou, kterou změří senzor HC-SR04. Senzor měří s určitou nepřesností, což je v porovnávání zahrnuto a měření vzdálenosti je provedeno třikrát a poté zprůměrováno. Kompletní výpis zdrojového kódu uzlu kontrola_vzdalenosti je přiložen v příloze B.

```
def zmer_vzdalenost(req):
    soucet = 0.0
    for i in range(3):
        GPIO.output(ultrazvuk_trig, 1)
        time.sleep(0.00001)
        GPIO.output(ultrazvuk_trig, 0)
        while GPIO.input(ultrazvuk_echo) == 0 :
            pulse_start_time = time.time()
        while GPIO.input(ultrazvuk_echo) == 1 :
            pulse_end_time = time.time()
        pulse_duration = pulse_end_time - pulse_start_time
```

```
        vzdalenost = pulse_duration * 171500
        soucet += vzdalenost
    if ( (req.ocekavana - 3) > (soucet / 3) > (req.ocekavana + 3) ):
        return vzdalenostResponse('NOK')
    else:
        return vzdalenostResponse('OK')
```

6.3.4.9 tlacitko

Posledním uzlem pro řízení modelu je uzel, čekající na stisk tlačítka. Uzel dostane pokyn a následně čeká, dokud nedojde ke stisku tlačítka. Ihned po stisku je odeslána odpověď uzlu driver. Kompletní výpis zdrojového kódu uzlu tlacitko je přiložen v příloze B.

```
def pockej_na_stisk(req):
    while GPIO.input(tlacitko) == 0:
        pass
    return stisknutiResponse('Stisknuto')
```

6.3.5 Soubory pro komunikaci

Pro komunikaci mezi uzlem driver a uzly ovladani_serva a led nebylo zapotřebí vytvářet žádný soubor pro přeposílání zpráv, jelikož zpráva je pouze jednoduchý řetězec. Proto byl pro odesílání zprávy použit soubor std_msgs.msg, který je součástí balíků ROS.

Komunikace mezi driverem a ostatními uzly probíhá pomocí služeb, takže bylo zapotřebí vytvořit soubory pro zasílání požadavků a odpovědí. Soubory jsou umístěny ve vytvořeném adresáři srv uvnitř adresáře r_ruka. Jedná se o soubory home.srv, polohovani.srv, objekt.srv, stisknuti.srv a vzdalenost.srv.

```
$ cd ~/bakalarka /catkin_ws/src/r_ruka
$ mkdir srv
$ cd srv
$ nano polohovani.srv
```

Příkladem je soubor polohovani.srv, kdy první část zprávy představuje požadavek, druhá část za třemi pomlčkami je odpovědí.

```
int64 nema
int64 krokac1
int64 krokac2
---
string vysledek
```

Následně je zapotřebí upravit soubory package.xml a CMakeList.txt, které byly automaticky vytvořeny při vytváření balíku r_ruka. V souboru package.xml je potřeba odkomentovat dva řádky.

```
<build_depend>message_generation</build_depend>
<exec_depend>message_runtime</exec_depend>
```

Do souboru CMakeList.txt je potřeba přidat názvy souborů použitých pro zasílání služeb.

```
add_service_files(
  FILES
  home.srv
  polohovani.srv
  objekt.srv
  stisknuti.srv
  vzdalenost.srv
)
```

6.3.6 Spouštěcí soubor

Pro spuštění každého uzlu je potřeba otevřít samostatné okno, přičemž s vyšším počtem uzlů je tento postup značně nepohodlný. Proto ROS nabízí možnost spustit všechny uzly naráz použitím spouštěcího souboru. Pro tento soubor je vytvořen adresář launch uvnitř adresáře r_ruka. V adresáři launch je vytvořen soubor r_ruka.launch, který zajistí spuštění všech uzlů najednou.

```
$ cd ~/bakalarka /catkin_ws/src/r_ruka
$ mkdir launch
$ cd launch
$ nano r_ruka.launch
```

Následně je potřeba soubor r_ruka.launch napsat podle stanovené šablony. První položkou každého řádku je balík, ve kterém se nachází zdrojový kód uzlu, další je název souboru obsahující tento kód a třetí položka je pojmenovaní uzlu v systému ROS.

```
<launch>

     <node pkg="r_ruka" type="led.py" name="led"/>
     <node pkg="r_ruka" type="ovladani_serva.py"
name="ovladani_serva"/>
     <node pkg="r_ruka" type="homing.py" name="homing"/>
     <node pkg="r_ruka" type="tlacitko.py" name="tlacitko"/>
     <node pkg="r_ruka" type="kontrola_predmetu.py"
name="kontrola_predmetu"/>
```

```
     <node pkg="r_ruka" type="kontrola_vzdalenosti.py"
name="kontrola_vzdalenosti"/>
     <node pkg="r_ruka" type="kontrola_uchopeni.py"
name="kontrola_uchopeni"/>
     <node pkg="r_ruka" type="pohyb.py" name="pohyb"/>
     <node pkg="r_ruka" type="driver.py" name="driver"/>

</launch>
```

6.3.7 Spuštění řídících uzlů

Před spuštěním je důležité nastavit všem uzlům potřebná práva. Příkladem je nastavení práv uzlu driver.

```
$ cd ~/bakalarka /catkin_ws/src/r_ruka/skripty
$ chmod u+x driver.py
```

Po nastavení práv všem uzlům a vytvoření potřebných souborů zmíněných výše, je nutné znovu použít příkaz catkin_make uvnitř adresáře catkin_ws. Tím dojde k automatickému vytvoření kódu pro zprávy a služby.

```
$ cd ~/bakalarka /catkin_ws
$ catkin_make
```

Nyní jsou spuštěny uzly ROS Master, Parametr Server a rosout, zmíněné v teoretické části.

```
$ roscore
```

Následně je potřeba otevřít nový terminál v programu PuTTY, ve kterém je po volání catkin_make nutné použít příkaz pro změnění souboru setup.bash. Následně je potřeba povolit přístup k rozhraní GPIO. Nakonec jsou pomocí spouštěcího souboru spuštěny všechny potřebné uzly a tím zahájeno řízení modelu.

```
$ cd ~/bakalarka /catkin_ws
$ source ./devel/setup.bash
$ sudo chmod a+rw /dev/gpiomem
$ roslaunch r_ruka r_ruka.launch
```

6.4 Testování

První spouštění uzlů probíhalo na několik pokusů, jelikož byly hlášeny syntaktické chyby v programech, které musely být opraveny. Následně bylo při prvním pokusu o najetí do základní polohy zjištěno, že se motor 28BYJ-48 pohybující druhým ramenem otáčí v opačném

směru než totožný motor pohybující ramenem třetím. Proto byla do uzlů homing a pohyb přidána obrácená sekvence pro krokování (sekvence2 viz. příloha B), která v programech původně nebyla a programy byly upraveny. Vzniklý problém by mohl být způsoben prohozením kabelů během jejich pájení ke konektorům, jelikož vadný ovladač autor vyloučil vyzkoušením jiného.

Po prvním najetí do základní pozice model ihned začal najíždět na pozici předmětu, což se opakovalo i při vypnutí a opětovném zapnutí uzlů. Problém byl způsoben lehkým kmitáním napětí na pinu načítajícím stav tlačítka, takže nebylo zajištěno nulové napětí ve chvíli, kdy tlačítko nebylo stisknuté. To bylo vyřešeno zapojením pull-down rezistoru k tomuto pinu (viz. příloha B) a tlačítko začalo fungovat správně.

Následovalo první najetí modelu na požadovanou pozici, na které se zastavil a v navazujících pohybech nepokračoval. Autor se domníval, že vzniklý problém by mohl být způsoben chybou ve zdrojových kódech uzlů kontrola_vzdalenosti nebo driver a po kontrole zjistil, že došlo k překlepu při psaní názvu služby v uzlu kontrola_vzdálenosti. To způsobilo, že uzel driver marně hledal požadovanou službu, čímž se zastavil cyklus chodu modelu. Po opravení překlepu se problém znovu nevyskytl.

Jako testovací předměty byly vytištěny kostičky o rozměrech 25 x 25 x 25 mm. Během prvního přenášení kostička z efektoru vypadla a tento problém se opakoval. Autor ho vyřešil nalepením textilní lepící pásky na ramena efektoru, čímž problém vyřešil. Kostička se ale občas v efektoru lehce naklonila, proto byla pro odkládání předmětu na jeho původní pozici upravena zetová souřadnice polohy, aby náhodou nedošlo ke kolizi předmětu s odkládací plochou.

Poslední obtíž objevující se během testování bylo občasné zaseknutí v nejnižší poloze prvního ramena při pohybu směrem vzhůru. Autor předpokládá, že problém byl způsoben lehkým ohnutím pohybového šroubu. Z toho důvodu se rozhodl odebrat spodní trapézovou matici z prvního ramena a povolit červíky pro upnutí pohybového šroubu v ložiskovém tělese. Problém se tím podařilo vyřešit a během následného testování se znovu neobjevil. Autor se však domnívá, že by se mohl při dlouhodobějším testování občasně vyskytovat a nejlepším řešením by bylo pořídit nový pohybový šroub.

Po vyřešení zmíněných problémů začal model fungovat tak, jak autor zamýšlel a během půlhodinového testovacího provozu se další problémy nevyskytly.

6.5 Možnosti využití v reálném světě

Sestrojený model by v reálném světě našel uplatnění jako manipulátor, který by si kontroloval stav pozic. Mohl by obsluze hlásit nedostatek přemisťovaných předmětů (polotovarů, obrobků), či problémy v úkonech navazujících na manipulaci v závislosti na obsazení odkládacích pozic.

Jelikož je ROS velice pružný a nahrazení uzlů v již zhotoveném systému nepřináší veliké obtíže, mohl by být reálný robot velice flexibilní. Možnost libovolně měnit čidla s případným doplněním modulárnosti efektoru by naskytovala přizpůsobení požadované situaci ve výrobě. Jednoduchou konfigurací spouštěcího souboru, případně použitím jiného, by stačilo spustit požadované uzly, a robotický systém by se choval jinak. Mohl by tedy v jednu chvíli pouze zakládat díly a v druhou už by mohl tyto díly kontrolovat např. použitím kamery a rozměrově nevyhovující díly odkládat stranou.

7 Závěr

Tato práce se zabývá celkovou stavbou robotického systému od konstrukčního návrhu, přes výběr potřebného hardwaru a softwaru až po naprogramování samotných řídících uzlů, přičemž je zde kladena podmínka na vykazování autonomního chování a řízení systému pomocí ROS.

Nejdříve byl navržen kinematický řetězec, který mohl být z technologií dostupných autorovi práce zrealizován. Především se jednalo o realizaci jednotlivých pohybů. Následně proběhl návrh konstrukce a byl sestrojen model z navržených dílů. Posléze byly z navržené struktury řetězce a rozměrů jednotlivých dílů robotické ruky vyjádřeny vzorce, které byly v další části práce využity pro řízení pohybu sestrojeného modelu.

Pro realizaci pohybu a řízení modelu bylo potřeba vybrat vhodné aktuátory, čidla a software. Výběr aktuátorů se podřizoval mechanické konstrukci a výběr čidel závisel na úkonu, který navržený model vykonává. Rozhodnutí o volbě softwaru bylo jednoznačné a volba se přizpůsobovala podmínce zadání, tedy řízení pomocí ROS.

Řízení modelu robotické ruky je rozděleno do devíti uzlů, přičemž každý z uzlů se stará o jednotlivou úlohu potřebnou pro správnou funkci celého systému. Řízení pomocí ROS nabízí modulárnost, takže je zde možnost vyzkoušet jiná čidla, případně jiné úkony vykonávané sestrojeným modelem, což je naznačeno během rozboru možností využití v reálném světě.

Všechny stanovené cíle práce se podařilo naplnit. Řízení robotické ruky pomocí ROS je rozebráno v teoretické části. Model se podařilo sestrojit, byl vybrán vhodný hardware a software pro jeho řízení a řízení bylo následně naprogramováno. Autonomní chování představuje rozhodování modelu o zjištěné situaci. Testování potvrdilo funkčnost sestrojeného modelu a úspěšné využití ROS k řízení modelu robotické ruky.

Literatura a použité zdroje

[1] ŠOLC, František a Luděk ŽALUD. Robotika. Brno, 2002. Dostupné také z: http://media1.wgz.cz/files/media1:5100dca52f8f1.pdf.upl/Robotika.pdf. Fakulta elektrotechniky a komunikačních technologií, Vysoké učení technické v Brně.

[2] SKAŘUPA, Jiří. Průmyslové roboty a manipulátory: učební text. Ostrava: Ediční středisko VŠB – TUO, 2007. ISBN 978-80-248-1522-0. Dostupné také z: http://www.elearn.vsb.cz/archivcd/FS/PRM/Text/Skripta_PRaM.pdf

[3] SKAŘUPA, Jiří. Roboty a manipulátory: Učební text. Ostrava: VŠB – Technická univerzita Ostrava, 2012. ISBN 978-80-248-2613-4. Dostupné také z: http://www.person.vsb.cz/archivcd/FS/RaMa/Roboty%20a%20manipulatory.pdf

[4] KÁRNÍK, Ladislav. Praktické aplikace servisních robotů: Studijní opora. Ostrava: Vysoká škola báňská – Technická univerzita Ostrava, 2011. ISBN 978-80-248-2727-8. Dostupné také z: http://projekty.fs.vsb.cz/147/ucebniopory/978-80-248-2727-8.pdf

[5] KÁRNÍK, Ladislav. Využití servisních robotů v nestrojírenských aplikacích: Studijní opora. Ostrava: Vysoká škola báňská – Technická univerzita Ostrava, 2010. ISBN 978-80-248-2728-5. Dostupné také z: http://projekty.fs.vsb.cz/147/ucebniopory/978-80-248-2728-5.pdf

[6] ISO 8373:2012: Robots and robotic devices — Vocabulary. 2. vydání. Geneva: International Organization for Standardization, 2012. ISO 8373:2012.

[7] KARGER, Adolf a Marie KARGEROVÁ. Základy robotiky a prostorové kinematiky. Praha: České vysoké učení technické, 2000. ISBN 80-010-2183-1.

[8] IRB 360 FlexPicker™. In: ABB [online]. Zurich: ABB Asea Brown Boveri, ©2019 [cit. 2019-03-21]. Dostupné z: https://new.abb.com/products/robotics/cs/prumyslove-roboty/irb-360

[9] GOUBEJ, Martin, Martin ŠVEJDA a Miloš SCHLEGEL. Úvod do mechatroniky, robotiky a systémů řízení pohybu: Skriptum pro studenty doktorských programů v oboru automatické řízení. Západočeská univerzita v Plzni, 2012. Dostupné také z: http://home.zcu.cz/~msvejda/URM/materialy/Uvod%20do%20mechatroniky.pdf

[10] IRB 2600. In: ABB [online]. Zurich: ABB Asea Brown Boveri, ©2019 [cit. 2019-03-21]. Dostupné z: https://new.abb.com/products/robotics/cs/prumyslove-roboty/irb-2600

[11] GREPL, Robert. Kinematika a dynamika mechatronických systémů. Brno: Akademické nakladatelství CERM, 2007. ISBN 978-80-214-3530-8.

[12] ROS.org | About ROS. ROS.org | Powering the world's robots [online]. Mountain View (California): Open Source Robotics Foundation [cit. 2019-03-21]. Dostupné z: http://www.ros.org/about-ros/

[13] ROS/Introduction - ROS Wiki. Documentation - ROS Wiki [online]. Mountain View (California): Open Source Robotics Foundation, 2018 [cit. 2019-03-21]. Dostupné z: http://wiki.ros.org/ROS/Introduction

[14] ROS 101: INTRO TO THE ROBOT OPERATING SYSTEM. Clearpath Robotics [online]. Kitchener (Canada): © Clearpath Robotics, 2014 [cit. 2019-03-21]. Dostupné z: https://www.clearpathrobotics.com/blog/2014/01/how-to-guide-ros-101/

[15] ROS/Tutorials/UnderstandingNodes - ROS Wiki. Documentation - ROS Wiki [online]. Mountain View (California): Open Source Robotics Foundation, 2018 [cit. 2019-03-21]. Dostupné z: http://wiki.ros.org/ROS/Tutorials/UnderstandingNodes

[16] Master - ROS Wiki. Documentation - ROS Wiki [online]. Mountain View (California): Open Source Robotics Foundation, 2018 [cit. 2019-03-21]. Dostupné z: http://wiki.ros.org/Master

[17] Roscore - ROS Wiki. Documentation - ROS Wiki [online]. Mountain View (California): Open Source Robotics Foundation, 2016 [cit. 2019-03-21]. Dostupné z: http://wiki.ros.org/roscore

[18] Nodes - ROS Wiki. Documentation - ROS Wiki [online]. Mountain View (California): Open Source Robotics Foundation, 2018 [cit. 2019-03-21]. Dostupné z: http://wiki.ros.org/Nodes

[19] Topics - ROS Wiki. Documentation - ROS Wiki [online]. Mountain View (California): Open Source Robotics Foundation, 2019 [cit. 2019-03-21]. Dostupné z: http://wiki.ros.org/Topics

[20] Services - ROS Wiki. Documentation - ROS Wiki [online]. Mountain View (California): Open Source Robotics Foundation, 2018 [cit. 2019-03-21]. Dostupné z: http://wiki.ros.org/Services

[21] Messages - ROS Wiki. Documentation - ROS Wiki [online]. Mountain View (California): Open Source Robotics Foundation, 2016 [cit. 2019-03-21]. Dostupné z: http://wiki.ros.org/Messages

[22] Msg - ROS Wiki. Documentation - ROS Wiki [online]. Mountain View (California): Open Source Robotics Foundation, 2019 [cit. 2019-03-21]. Dostupné z: http://wiki.ros.org/msg

[23] Srv - ROS Wiki. Documentation - ROS Wiki [online]. Mountain View (California): Open Source Robotics Foundation, 2017 [cit. 2019-03-21]. Dostupné z: http://wiki.ros.org/srv

[24] Distribution - ROS Wiki. Documentation - ROS Wiki [online]. Mountain View (California): Open Source Robotics Foundation, 2018 [cit. 2019-03-21]. Dostupné z: http://wiki.ros.org/Distributions

[25] Kinetic - ROS Wiki. Documentation - ROS Wiki [online]. Mountain View (California): Open Source Robotics Foundation, 2018 [cit. 2019-03-21]. Dostupné z: http://wiki.ros.org/kinetic

[26] The MagPi: The official Raspberry Pi magazine. 2016, (43). Dostupné také z: https://www.raspberrypi.org/magpi/issues/43/

[27] Raspberry Pi 3 Model B. Raspberry Pi: Teach, Learn, and Make with Raspberry Pi [online]. Cambridge: Raspberry Pi Foundation [cit. 2019-03-21]. Dostupné z: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/

[28] Microcon 2015: Kompletní pohony s krokovými motry. Praha: © Microcon, 2015. Dostupné také z: https://ecitydoc.com/download/wwwmicroconcz_pdf

[29] A4988: DMOS Microstepping Driver with Translator and Overcurrent Protection. Worcester (Massachusetts): Allegro MicroSystems, ©2016. Dostupné také z: https://www.allegromicro.com/en/Products/Motor-Driver-And-Interface-ICs/Bipolar-Stepper-Motor-Drivers.aspx

[30] 28BYJ-48 Stepper Motor Pinout Wiring, Specifications, Uses Guide & Datasheet. Omponents101 - Electronic Components Pinouts, Details & Datasheets [online]. Components101, ©2019 [cit. 2019-03-21]. Dostupné z: https://components101.com/motors/28byj-48-stepper-motor

[31] Servo Motor 4,3 g pro Aeromodelling MiNi Micro. Arduino-shop.cz: VELKOOBCHOD, MALOOBCHOD S ARDUINEM [online]. © Copyright ECLIPSERA [cit. 2019-03-21]. Dostupné z: https://arduino-shop.cz/arduino/1577-mini-micro-servo-motor-pro-aeromodelling-4-3-g-1489340899.html

[32] Infračervený senzor překážek. Arduino-shop.cz: VELKOOBCHOD, MALOOBCHOD S ARDUINEM [online]. © Copyright ECLIPSERA [cit. 2019-03-21]. Dostupné z: https://arduino-shop.cz/arduino/3086-infracerveny-senzor-prekazek.html

[33] HC-SR04 User Guide. Elecfreaks Store, ©2011-2017. Dostupné také z: https://www.elecfreaks.com/estore/hc-sr04-ultrasonic-sensor-distance-measuring-module-ultra01.html

[34] LRS-50 series: 50W Single Output Switching Power Supply. MEAN WELL Enterprises Co., 2017. Dostupné také z: https://www.meanwell.com/productPdf.aspx?i=399

[35] Step up/down modul XL6009. Verze 1.0. ECLIPSERA, 2017. Dostupné také z: https://arduino-shop.cz/docs/produkty/0/373/1502436407.pdf

[36] LM2596 SIMPLE SWITCHER® Power Converter 150-kHz 3-A Step-Down Voltage Regulator. Dallas, Texas: Texas Instruments Incorporated, 2016. Dostupné také z: http://www.ti.com/lit/ds/symlink/lm2596.pdf

[37] About the Ubuntu project | Ubuntu. The leading operating system for PCs, IoT devices, servers and the cloud | Ubuntu [online]. Canonical, ©2019 [cit. 2019-03-21]. Dostupné z: https://www.ubuntu.com/about

[38] Ubuntu for IoT Developers documentation. Ubuntu documentation [online]. Canonical, ©2019 [cit. 2019-03-21]. Dostupné z: https://docs.ubuntu.com/core/en/

[39] History and License — Python 3.7.3rc1 documentation. Python.org [online]. Python Software Foundation, ©2001-2019 [cit. 2019-03-21]. Dostupné z: https://docs.python.org/3/license.html

[40] The Python Tutorial — Python 3.7.3rc1 documentation. Python.org [online]. Python Software Foundation, ©2001-2019 [cit. 2019-03-21]. Dostupné z: https://docs.python.org/3/tutorial/

[41] 1. Whetting Your Appetite — Python 3.7.3rc1 documentation. Python.org [online]. Python Software Foundation, ©2001-2019 [cit. 2019-03-21]. Dostupné z: https://docs.python.org/3/tutorial/appetite.html

[42] UPTON, Eben a Gareth HALFACREE. Raspberry Pi: uživatelská příručka. 2., aktualizované vydání. Přeložil Jakub Goner. Brno: Computer Press, 2016. ISBN 978-802-5148-198.

[43] RPi.GPIO · PyPi. PyPi - Python Package Index · PyPi [online]. Python Software Foundation, ©2019 [cit. 2019-03-21]. Dostupné z: https://pypi.org/project/RPi.GPIO/

[44] 9.2. math — Mathematical functions — Python 2.7.16 documentation. Python.org [online]. Python Software Foundation, 2019 [cit. 2019-03-21]. Dostupné z: https://docs.python.org/2/library/math.html

[45] 15.3. time — Time access and conversions — Python 2.7.16 documentation. Python.org [online]. Python Software Foundation, 2019 [cit. 2019-03-21]. Dostupné z: https://docs.python.org/2/library/time.html

Seznam zkratek

API	Application Programming Interface
GPIO	General-purpose input/output
HDMI	High-Definition Multimedia Interface
IP	Internet Protocol
LED	Light Emitting Diode
LTS	Long Term Support
PWM	Pulse Width Modulation
R	Rotační vazba
ROS	Robot Operating System
SR	Sériový Robot
SSH	Secure Shell
T	Translační vazba
USB	Universal Serial Bus

A Seznam dílů modelu

Název dílu	Počet kusů
Dřevěná deska 50 x 50 x 1,9 mm	1
Spodní díl základny	1
Horní díl základny	1
Vodící tyč	3
Horní díl prvního ramena	1
Spodní díl prvního ramena	1
Distanční sloupek 44 mm	4
Díl druhého ramena	2
Distanční sloupek 22 mm	6
Horní díl třetího ramena	1
Spodní díl třetího ramena	1
Nosný díl efektoru	1
Pohyblivé rameno efektoru	1
Pevné rameno efektoru	1
Držák čidla pro kontrolu uchopení	1
Držák koncového spínače prvního ramena	1
Držák koncového spínače druhého a třetího ramene	1
Velká krabička na elektroniku	1
Přední panel velké krabičky	1
Patro velké krabičky	1
Víko velké krabičky	1
Menší krabička na tlačítko a led modul	1
Víko menší krabičky	1

Tabulka A.1: Díly vytištěné nebo vyrobené autorem práce

Název dílu, počet kusů	**Cena**	**Zakoupeno v:**
Ložisko 6807-2RS, 4 ks	290,00 Kč	CRAVT s.r.o.
Ložisko LM8UU-MTM, 4 ks	144,00 Kč	https://www.materialpro3d.cz
Ložiskové těleso KFL08, 1 ks	49,00 Kč	https://www.postavrobota.cz
Trapézový šroub 200 mm d8T8, 1 ks	99,00 Kč	https://www.postavrobota.cz
Trapézová matice d8T8, 2 ks	78,00 Kč	https://www.postavrobota.cz
Pružná spojka 5x8x25, 1 ks	90,00 Kč	https://www.materialpro3d.cz
Jehlové ložisko 3 x 6,5 x 6 mm, 1 ks	67,75 Kč	https://www.arkov.cz
Pojistný kroužek na hřídel 14 mm, 8 ks	4,87 Kč	https://www.arkov.cz
Matice M8 vysoká, 9 ks	4,24 Kč	https://www.arkov.cz
Matice M8 nízká, 3 ks	1,32 Kč	https://www.arkov.cz
Podložka 8 mm, 12 ks	7,15 Kč	https://www.arkov.cz
Šroub M5x20, 2 ks	1,50 Kč	https://www.arkov.cz
Matice M5, 2 ks	0,71 Kč	https://www.arkov.cz
Podložka 5 mm, 2 ks	1,67 Kč	https://www.arkov.cz
Šroub M3x10, 24 ks	5,57 Kč	https://www.arkov.cz
Šroub M3x16, 16 ks	8,60 Kč	https://www.arkov.cz
Šroub M3x40, 3 ks	1,38 Kč	https://www.arkov.cz
Matice M3, 10 ks	1,00 Kč	https://www.arkov.cz
Podložka 3 mm, 8 ks	0,96 Kč	https://www.arkov.cz
Šroub do plechu 2,9x6,5; 34 ks	5,20 Kč	https://www.arkov.cz
Celková cena:	**862,92 Kč**	

Tabulka A.2: Zakoupené díly mechanické konstrukce

Název, počet kusů	**Cena**	**Zakoupeno v:**
Raspberry Pi 3 Model B + příslušenství, 1 ks	1 540,00 Kč	http://rpishop.cz
SX17-1005VLQCEF, 1 ks	350,00 Kč	https://www.materialpro3d.cz
A4988, 1 ks	78,00 Kč	KONTAKT ELEKTRONIK spol. s.r.o.
28BYJ-48 + ovladač, 2 ks	184,00 Kč	https://arduino-shop.cz
GH-S43A, 1 ks	129,60 Kč	https://arduino-shop.cz
FC-51, 2 ks	130,00 Kč	KONTAKT ELEKTRONIK spol. s.r.o.
HC-SR04, 1 ks	89,00 Kč	KONTAKT ELEKTRONIK spol. s.r.o.
Koncový spínač, 3 ks	90,00 Kč	https://laskarduino.cz
LRS-50-12	292,00 Kč	https://www.gme.cz
DC-DC XL6009 step up/down modul, 1 ks	99,00 Kč	KONTAKT ELEKTRONIK spol. s.r.o.
DC-DC step-down měnič s LM2596, 1 ks	83,00 Kč	https://arduino-shop.cz
Tlačítko, 1 ks	15,00 Kč	KONTAKT ELEKTRONIK spol. s.r.o.
KY-16, 1 ks	28,00 Kč	https://laskarduino.cz
5 x nulovací můstek	80,55 Kč	K & V ELEKTRO a.s.
DIN panelová zásuvka 7 pinů, 1 ks	30,00 Kč	KONTAKT ELEKTRONIK spol. s.r.o.
DIN vidlice 7 pinů, 1 ks	20,00 Kč	KONTAKT ELEKTRONIK spol. s.r.o.
DIN panelová zásuvka 5 pinů, 5 ks	80,00 Kč	KONTAKT ELEKTRONIK spol. s.r.o.
DIN vidlice 5 pinů, 5 ks	45,00 Kč	KONTAKT ELEKTRONIK spol. s.r.o.
Propojovací kabely	cca 60 Kč	
Celková cena:	3 423,15 Kč	

Tabulka A.3: Hardware použitý pro model

B Kompletní výpis zdrojových kódů

driver

```
#!/usr/bin/env python

import rospy
import time
import math
from sys import exit
from std_msgs.msg import String
from r_ruka.srv import *

#délky ramen, velikosti kroků (u SX17 je krok přepočten na lineární
#posun, v případě 28BYJ-48 je to otočení po provedení 8krokové
#sekvence) [mm, °]
l1 = 71.0
l2 = 149.0
x4_cidlo = 122.15
x4_ee = 183.25
y4_cidlo = -58.0
krok_krokace12 = 0.7031525
posun_na_krok = 0.04
delay = 0.5
#souřadnice jednotlivých poloh [x, y, z], y je zadáno po přepočtení
#ze systému 0; vzdálenosti pro kontrolu
poloha_home = [90.13, -100.13, 0.0]
poloha_predmetu_cidlo = [-115.0, -90.0, -125.52]
poloha_predmetu_polozeni = [-115.0, -90.0, -121.52]
poloha_nad_predmetem_cidlo = [-115.0, -90.0, -97.0]
poloha_odkladani1 = [190.0, -20.0, -125.52]
poloha_nad1 = [190.0, -20.0, -100.0]
poloha_odkladani2 = [180.0, 130.0, -125.52]
poloha_nad2 = [180.0, 130.0, -100.0]
vzdalenost_predmet = 175.0
vzdalenost_nad_predmet = 151.0
vzdalenost_odkladani = 179.0
vzdalenost_nad = 154.0

#výpočet přepony
def prepona(x, y):
    p = math.sqrt((x**2) + (y**2))
    return p
#výpočet úhlu theta3
def theta3(prepona, rameno_2):
    theta_3 = math.degrees(math.acos( ((l2**2) + (rameno_2**2) -
(prepona**2)) / (2*l2*rameno_2) ))
```

```
    return theta_3
#výpočet úhlu theta2, ošetření pro úhel rostoucí v záporném směru
def theta2(x, y, rameno_2):
    if (((x - 149)**2+y**2 <= rameno_2**2) or ((x >= 0)and(y<= 0))):
          theta_2 = - (180 - theta3(prepona(x, y), rameno_2))
    else:
          theta_2 = 180 - theta3(prepona(x, y), rameno_2)
    return theta_2
#výpočet úhlu theta1, ošetření pro jednotlivé podmínky
def theta1(x, y, rameno_2):
    theta_3 = theta3(prepona(x, y), rameno_2)
    uhel_xy = float
    if ((x - 149)**2 + y**2 <= rameno_2**2):
          uhel_xy = math.degrees(math.atan(y/x))
          theta_1 = (uhel_xy + math.degrees(math.asin
(rameno_2/prepona(x, y)*math.sin(math.radians(theta_3)))) )
    elif ( (x > 0) and (y < 0) ):
          y = -y
          uhel_xy = math.degrees(math.atan(y/x))
          theta_1 = ( math.degrees(math.asin ( rameno_2/prepona(x,
y)*math.sin( math.radians(theta_3)))) - uhel_xy )
    else:
          if (x >= 0 and y >=0):
                uhel_xy = math.degrees(math.atan(y/x))
          elif (x <= 0 and y >=0):
                x = -x
                uhel_xy = (180 - math.degrees(math.atan(y/x)))
          else:
                x = -x
                y = -y
                uhel_xy = (180 + math.degrees(math.atan(y/x)))
          theta_1 = ( uhel_xy - math.degrees(math.asin
(rameno_2/prepona(x, y)*math.sin(math.radians(theta_3)))) )
    return theta_1 - 90
#výpočet nových souřadnic x a y z vypočtených úhlů polohy čidla
def vypocet_xy(theta_1, theta_2):
    x = ( -x4_cidlo*math.sin(math.radians(theta_1 + theta_2)) -
y4_cidlo*math.cos(math.radians(theta_1 + theta_2)) -
l2*math.sin(math.radians(theta_1)) )
    y = ( x4_cidlo*math.cos(math.radians(theta_1 + theta_2)) -
y4_cidlo*math.sin(math.radians(theta_1 + theta_2)) +
l2*math.cos(math.radians(theta_1)) )
    poloha_xy = [x, y]
    return poloha_xy

def prepocet_cidlo_endeffector(poloha):
    theta_1 = theta1(poloha[0], poloha[1], x4_cidlo)
    theta_2 = theta2(poloha[0], poloha[1], x4_cidlo)
    nove = vypocet_xy(theta_1, theta_2)
```

```
    xyz = [nove[0], nove[1], poloha[2]]
    return xyz
#výpočet kroků na zadanou pozici
def pocty_kroku_na_pozici(poloha, rameno_2):
    theta_1 = theta1(poloha[0], poloha[1], rameno_2)
    theta_2 = theta2(poloha[0], poloha[1], rameno_2)
    kroky_nema =  int(round(poloha[2]/posun_na_krok))
    kroky_prvniho = int(round(theta_1 / krok_krokace12))
    kroky_druheho = int(round(theta_2 / krok_krokace12))
    vypoctene_kroky = [kroky_nema, kroky_prvniho, kroky_druheho]
    return vypoctene_kroky
#funkce pro napočítání kroků pro jednotlivé motory
def kroky_pro_motory(stara_poloha, nova_poloha, rameno_stare,
rameno_nove):
    kroky_stare_polohy = pocty_kroku_na_pozici(stara_poloha,
rameno_stare)
    kroky_nove_polohy = pocty_kroku_na_pozici(nova_poloha,
rameno_nove)
    kroky_nema =  kroky_nove_polohy[0] - kroky_stare_polohy[0]
    kroky_prvniho = kroky_nove_polohy[1] - kroky_stare_polohy[1]
    kroky_druheho = kroky_nove_polohy[2] - kroky_stare_polohy[2]
    kroky_motoru = [kroky_nema, kroky_prvniho, kroky_druheho]
    return kroky_motoru
#funkce volající service polohovani
def najeti_client(pocty_kroku):
    rospy.wait_for_service('polohovani')
    try:
          najed = rospy.ServiceProxy('polohovani', polohovani)
          resp=najed(pocty_kroku[0], pocty_kroku[1], pocty_kroku[2])
          return resp.vysledek
    except rospy.ServiceException, e:
          return 'chyba'
#funkce volající service homovani
def homing_client(info):
    rospy.wait_for_service('homovani')
    try:
          nahomuj = rospy.ServiceProxy('homovani', home)
          resp = nahomuj(info)
          return resp.vysledek
    except rospy.ServiceException, e:
          return 'chyba'
#funkce volající service zjisteni
def zjisti_client():
    rospy.wait_for_service('zjisteni')
    try:
          zjisti = rospy.ServiceProxy('zjisteni', objekt)
          resp = zjisti('dej_info')
          return resp.vysledek
    except rospy.ServiceException, e:
```

```
            return 'chyba'
#funkce volající service kontrola_uchopeni
def over_uchopeni_client():
    rospy.wait_for_service('kontrola_uchopeni')
    try:
            zjisti = rospy.ServiceProxy('kontrola_uchopeni', objekt)
            resp = zjisti('dej_info')
            return resp.vysledek
    except rospy.ServiceException, e:
            return 'chyba'
#funkce volající service stisk
def tlacitko_client():
    rospy.wait_for_service('stisk')
    try:
            tl = rospy.ServiceProxy('stisk', stisknuti)
            resp = tl('dej_info')
            return resp.vysledek
    except rospy.ServiceException, e:
            return 'chyba'
#funkce volající service porovnani_vzdalenosti
def vzdalenost_client(ocekavana):
    rospy.wait_for_service('porovnani_vzdalenosti')
    try:
            vzd=rospy.ServiceProxy('porovnani_vzdalenosti',vzdalenost)
            resp = vzd(ocekavana)
            return resp.porovnani
    except rospy.ServiceException, e:
            return 'chyba'
#funkce publikující data k tématu info_servu
def publisher_servo(info_pro_servo):
    servo_pub.publish(info_pro_servo)
#funkce publikující data k tématu info_led
def publisher_ledka(info_pro_ledku):
    led_pub.publish(info_pro_ledku)
#funkce pro volání vzdalenost_client a kontrolu výsledku
def over_vzdalenost(ocekavana):
    overeni = vzdalenost_client(ocekavana)
    if overeni == 'OK':
            pass
    else:
            publisher_ledka('cervena_blik')
            exit()
#funkce kontrolující výsledek, chyba=>rozblikání led+konec programu
def kontrola_vysledku(vysledek):
    if vysledek == 'chyba':
            publisher_ledka('cervena_blik')
            exit()
#funkce volaná při chybě v komunikaci, chyba => jako předchozí fce
def chyba():
```

```
    publisher_ledka('cervena_blik')
    exit()

if __name__ == "__main__":
    try:
#inicializace uzlu jako publishera v ROS
        servo_pub=rospy.Publisher('info_servu',String,queue_size=1)
        led_pub = rospy.Publisher('info_led', String, queue_size=1)
        rospy.init_node('driver', anonymous = True)
#najetí do základní pozice a rozvícení led do zelena
        kontrola_vysledku( homing_client('random') )
        publisher_ledka('zelena')
#hlavní cyklus řízení
        while (1):
#najetí na pozici předmětu efektorem (je prázdný)
            stara_poloha = poloha_home
            nova_poloha =
prepocet_cidlo_endeffector(poloha_predmetu_cidlo)
            kontrola_vysledku( tlacitko_client() )
            publisher_ledka('modra')
            kontrola_vysledku(
najeti_client(kroky_pro_motory(stara_poloha, nova_poloha, x4_cidlo,
x4_ee)) )
            over_vzdalenost(vzdalenost_odkladani)
            time.sleep(delay)
            stara_poloha = nova_poloha
            vysledek = over_uchopeni_client()
#kontrola, zda je na pozici předmět
            if vysledek == 'neuchopeno':
                nova_poloha =
prepocet_cidlo_endeffector(poloha_nad_predmetem_cidlo)
            elif vysledek == 'chyba':
                chyba()
            else:
                publisher_ledka('cervena')
                kontrola_vysledku( homing_client('pozice') )
                continue
#uchopení a kontrola uchopení
            publisher_servo('uchop')
            time.sleep(delay)
            kontrola_vysledku(
najeti_client(kroky_pro_motory(stara_poloha, nova_poloha, x4_ee,
x4_ee)))
            over_vzdalenost(vzdalenost_nad_predmet)
            stara_poloha = nova_poloha
            vysledek = over_uchopeni_client()
            if vysledek == 'neuchopeno':
                nova_poloha = poloha_odkladani1
            elif vysledek == 'chyba':
```

```
                    chyba()
               else:
                    publisher_servo('pust')
                    publisher_ledka('neuchopeno')
                    kontrola_vysledku( homing_client('pozice') )
                    continue
#přejetí na první pozici pro odložení, kontrola obsazení pozice a
#případné odložení a najetí do základní pozice
               kontrola_vysledku(
najeti_client(kroky_pro_motory(stara_poloha, nova_poloha, x4_ee,
x4_cidlo)) )
               over_vzdalenost(vzdalenost_odkladani)
               stara_poloha = nova_poloha
               vysledek = zjisti_client()
               if vysledek == 'obsazeno':
                    nova_poloha = poloha_nad2
               elif vysledek == 'chyba':
                    chyba()
               else:
                    nova_poloha =
prepocet_cidlo_endeffector(poloha_odkladani1)
                    kontrola_vysledku(
najeti_client(kroky_pro_motory(stara_poloha, nova_poloha, x4_cidlo,
x4_ee)) )
                    time.sleep(delay)
                    publisher_servo('pust')
                    time.sleep(delay)
                    stara_poloha = nova_poloha
                    nova_poloha = poloha_nad1
                    kontrola_vysledku(
najeti_client(kroky_pro_motory(stara_poloha, nova_poloha, x4_ee,
x4_cidlo)) )
                    kontrola_vysledku( homing_client('pozice') )
                    publisher_ledka('zelena')
                    continue
#přejetí na druhou pozici pro odložení, kontrola obsazení pozice a
#případné odložení a najetí do základní pozice
               kontrola_vysledku(
najeti_client(kroky_pro_motory(stara_poloha, nova_poloha, x4_cidlo,
x4_cidlo)) )
               over_vzdalenost(vzdalenost_nad)
               stara_poloha = nova_poloha
               nova_poloha = poloha_odkladani2
               kontrola_vysledku(
najeti_client(kroky_pro_motory(stara_poloha, nova_poloha, x4_cidlo,
x4_cidlo)) )
               over_vzdalenost(vzdalenost_odkladani)
               stara_poloha = nova_poloha
               vysledek = zjisti_client()
```

```
                if vysledek == 'obsazeno':
                        nova_poloha = poloha_nad_predmetem_cidlo
                elif vysledek == 'chyba':
                        chyba()
                else:
                        nova_poloha =
prepocet_cidlo_endeffector(poloha_odkladani2)
                        kontrola_vysledku(
najeti_client(kroky_pro_motory(stara_poloha, nova_poloha, x4_cidlo,
x4_ee)) )
                        time.sleep(delay)
                        publisher_servo('pust')
                        time.sleep(delay)
                        kontrola_vysledku( homing_client('pozice') )
                        publisher_ledka('zelena')
                        continue
#přejetí na pozici předmětu, kontrola obsazení pozice a případné
#odložení a najetí do základní pozice
                kontrola_vysledku(
najeti_client(kroky_pro_motory(stara_poloha, nova_poloha, x4_cidlo,
x4_cidlo)) )
                over_vzdalenost(vzdalenost_nad_predmet)
                stara_poloha = nova_poloha
                nova_poloha = poloha_predmetu_cidlo
                kontrola_vysledku(
najeti_client(kroky_pro_motory(stara_poloha, nova_poloha, x4_cidlo,
x4_cidlo)) )
                over_vzdalenost(vzdalenost_odkladani)
                stara_poloha = nova_poloha
                vysledek = zjisti_client()
                if vysledek == 'obsazeno':
                        publisher_ledka('cervena')
                        kontrola_vysledku( homing_client('pozice') )
#zjištěno, že jsou všechny pozice obsazeny ->kontrolní cyklus
                        while (1):
#postupné projetí pozic pro odkládání a pozice předmětu, jakmile
#objeví prázdnou pozici, odloží předmět a najede do základní pozice
                                kontrola_vysledku( tlacitko_client() )
                                publisher_ledka('modra')
                                stara_poloha = poloha_home
                                nova_poloha = poloha_odkladani1
                                kontrola_vysledku(
najeti_client(kroky_pro_motory(stara_poloha, nova_poloha, x4_cidlo,
x4_cidlo)) )
                                over_vzdalenost(vzdalenost_odkladani)
                                stara_poloha = nova_poloha
                                vysledek = zjisti_client()
                                if vysledek == 'obsazeno':
                                        nova_poloha = poloha_nad2
```

```
                        elif vysledek == 'chyba':
                              chyba()
                        else:
                              nova_poloha =
prepocet_cidlo_endeffector(poloha_odkladani1)
                              kontrola_vysledku(
najeti_client(kroky_pro_motory(stara_poloha, nova_poloha, x4_cidlo,
x4_ee)) )
                              time.sleep(delay)
                              publisher_servo('pust')
                              time.sleep(delay)
                              stara_poloha = nova_poloha
                              nova_poloha = poloha_nad1
                              kontrola_vysledku(
najeti_client(kroky_pro_motory(stara_poloha, nova_poloha, x4_ee,
x4_cidlo)) )
                              kontrola_vysledku(
homing_client('pozice') )
                              publisher_ledka('zelena')
                              break

                        kontrola_vysledku(
najeti_client(kroky_pro_motory(stara_poloha, nova_poloha, x4_cidlo,
x4_cidlo)) )
                        over_vzdalenost(vzdalenost_nad)
                        stara_poloha = nova_poloha
                        nova_poloha = poloha_odkladani2
                        kontrola_vysledku(
najeti_client(kroky_pro_motory(stara_poloha, nova_poloha, x4_cidlo,
x4_cidlo)) )
                        over_vzdalenost(vzdalenost_odkladani)
                        stara_poloha = nova_poloha
                        vysledek = zjisti_client()
                        if vysledek == 'obsazeno':
                              nova_poloha =
poloha_nad_predmetem_cidlo
                        elif vysledek == 'chyba':
                              chyba()
                        else:
                              nova_poloha =
prepocet_cidlo_endeffector(poloha_odkladani2)
                              kontrola_vysledku(
najeti_client(kroky_pro_motory(stara_poloha, nova_poloha, x4_cidlo,
x4_ee)) )
                              time.sleep(delay)
                              publisher_servo('pust')
                              time.sleep(delay)
                              kontrola_vysledku(
homing_client('pozice') )
```

```
                                    publisher_ledka('zelena')
                                    break

                              kontrola_vysledku(
najeti_client(kroky_pro_motory(stara_poloha, nova_poloha, x4_cidlo,
x4_cidlo)) )
                              over_vzdalenost(vzdalenost_nad_predmet)
                              stara_poloha = nova_poloha
                              nova_poloha = poloha_predmetu_cidlo
                              kontrola_vysledku(
najeti_client(kroky_pro_motory(stara_poloha, nova_poloha, x4_cidlo,
x4_cidlo)) )
                              over_vzdalenost(vzdalenost_odkladani)
                              stara_poloha = nova_poloha
                              vysledek = zjisti_client()
                              if vysledek == 'obsazeno':
                                    publisher_ledka('cervena')
                                    kontrola_vysledku(
homing_client('pozice') )
                                    continue
                              elif vysledek == 'chyba':
                                    chyba()
                              else:
                                    nova_poloha =
prepocet_cidlo_endeffector(poloha_predmetu_polozeni)
                                    kontrola_vysledku(
najeti_client(kroky_pro_motory(stara_poloha, nova_poloha, x4_cidlo,
x4_ee)) )
                                    time.sleep(delay)
                                    publisher_servo('pust')
                                    time.sleep(delay)
                                    kontrola_vysledku(
homing_client('pozice') )
                                    publisher_ledka('cervena')
                                    break
#chyba při komunikaci, nebo je pozice volná
                        elif vysledek == 'chyba':
                              chyba()
                        else:
                              nova_poloha =
prepocet_cidlo_endeffector(poloha_predmetu_polozeni)
                              kontrola_vysledku(
najeti_client(kroky_pro_motory(stara_poloha, nova_poloha, x4_cidlo,
x4_ee)) )
                              time.sleep(delay)
                              publisher_servo('pust')
                              time.sleep(delay)
                              kontrola_vysledku( homing_client('pozice') )
                              publisher_ledka('cervena')
```

```
    except KeyboardInterrupt:
        exit()
```

ovladani_serva.py

```
#!/usr/bin/env python

import rospy
import RPi.GPIO as GPIO
from std_msgs.msg import String

servo_pin = 18
#funkce pro nastaveni rozhraní GPIO
def nastav_GPIO():
    GPIO.setmode(GPIO.BCM)
    GPIO.setwarnings(False)
    GPIO.setup(servo_pin, GPIO.OUT)
#funkce pro ovládání serva
def uchop():
    servo.ChangeDutyCycle(7.7)

def pust():
    servo.ChangeDutyCycle(6.4)

def ovladani(data):
    if data.data == 'uchop':
        uchop()
    else :
        pust()
#funkce obsahující definici subscribera v ROS
def subscriber_servo():
    rospy.init_node('ovladani_serva', anonymous = True)
    rospy.Subscriber('info_servu', String, ovladani)
    rospy.spin()

if __name__ == '__main__' :
    nastav_GPIO()
    servo = GPIO.PWM(servo_pin, 50)
    servo.start(6.4)
    try:
        subscriber_servo()
    except KeyboardInterrupt:
        GPIO.cleanup()
        exit()
```

led.py

```
#!/usr/bin/env python

import rospy
import RPi.GPIO as GPIO
import time
from std_msgs.msg import String

R = 11
G = 9
B = 10
#funkce pro nastaveni rozhraní GPIO
def nastav_GPIO():
    GPIO.setmode(GPIO.BCM)
    GPIO.setwarnings(False)
    GPIO.setup(R, GPIO.OUT)
    GPIO.setup(G, GPIO.OUT)
    GPIO.setup(B, GPIO.OUT)
    GPIO.output(R, 0)
    GPIO.output(G, 0)
    GPIO.output(B, 0)
#funkce pro ovládání diody
def barva(data):
    GPIO.output(R, 0)
    GPIO.output(G, 0)
    GPIO.output(B, 0)
    if data.data == 'zelena':
          GPIO.output(G, 1)
    elif data.data == 'modra':
          GPIO.output(B, 1)
    elif data.data == 'cervena_blik':
          while (1):
                GPIO.output(R, 1)
                time.sleep(1)
                GPIO.output(R, 0)
                time.sleep(1)
    elif data.data == 'neuchopeno':
          while (1):
                GPIO.output(R, 1)
                GPIO.output(G, 1)
                GPIO.output(B, 1)
    else:
          GPIO.output(R, 1)
#funkce obsahující definici subscribera v ROS
def subscriber_led():
    rospy.init_node('led', anonymous = True)
    rospy.Subscriber('info_led', String, barva)
    rospy.spin()
```

```
if __name__ == '__main__' :
    nastav_GPIO()
    try:
          subscriber_led()
    except KeyboardInterrupt:
          GPIO.cleanup()
          exit()
```

pohyb.py

```
#!/usr/bin/env python

import rospy
import RPi.GPIO as GPIO
import time
from r_ruka.srv import *

#deklarace hodnot pinů motorů a časových rozestupů mezi kroky
#motorů
direction = 26
step = 19
sleep = 13
krokac1 = [4, 17, 27, 22]
krokac2 = [12, 16, 20, 21]
delay_nema = 0.00101
delay_krokace = 0.002
#sekvence pro krokování motoru druhého ramena směrem ke koncovým
#spínačům
sekvence1 = range(0, 8)
sekvence1[0] = [1,0,0,1]
sekvence1[1] = [0,0,0,1]
sekvence1[2] = [0,0,1,1]
sekvence1[3] = [0,0,1,0]
sekvence1[4] = [0,1,1,0]
sekvence1[5] = [0,1,0,0]
sekvence1[6] = [1,1,0,0]
sekvence1[7] = [1,0,0,0]
#sekvence pro krokování motoru třetího ramena směrem ke koncovým
#spínačům (obrácená sekvence1)
sekvence2 = range(0, 8)
sekvence2[0] = [1,0,0,0]
sekvence2[1] = [1,1,0,0]
sekvence2[2] = [0,1,0,0]
sekvence2[3] = [0,1,1,0]
sekvence2[4] = [0,0,1,0]
sekvence2[5] = [0,0,1,1]
sekvence2[6] = [0,0,0,1]
sekvence2[7] = [1,0,0,1]
```

```
#funkce pro nastaveni rozhraní GPIO
def nastav_GPIO():
    GPIO.setmode(GPIO.BCM)
    GPIO.setwarnings(False)
    GPIO.setup(direction, GPIO.OUT)
    GPIO.output(direction, 1)
    GPIO.setup(step, GPIO.OUT)
    GPIO.output(step,0)
    GPIO.setup(sleep, GPIO.OUT)
    GPIO.output(sleep, 0)
    for pin in range(4):
          GPIO.setup(krokac1[pin],GPIO.OUT)
          GPIO.output(krokac1[pin], 0)
          GPIO.setup(krokac2[pin],GPIO.OUT)
          GPIO.output(krokac2[pin], 0)
#funkce nastavující hodnoty na pinech pro provedení kroku motoru
#druhého ramena
def krok_krokac1(p1, p2, p3, p4):
    GPIO.output(krokac1[0], p1)
    GPIO.output(krokac1[1], p2)
    GPIO.output(krokac1[2], p3)
    GPIO.output(krokac1[3], p4)
#funkce nastavující hodnoty na pinech pro provedení kroku motoru
#třetího ramena
def krok_krokac2(p1, p2, p3, p4):
    GPIO.output(krokac2[0], p1)
    GPIO.output(krokac2[1], p2)
    GPIO.output(krokac2[2], p3)
    GPIO.output(krokac2[3], p4)
#funkce pro otočení motorů o požadovaný počet kroků
def najeti_do_pozice(kroky_nema, kroky_krokac1, kroky_krokac2):
#počet kroků motoru SX17 je kladný -> nejdříve motor SX17, pak
#pohyb motorů 28BYJ-48
    if (kroky_nema >= 0):
          GPIO.output(direction, 1)
          time.sleep(0.005)
          for i in range (kroky_nema):
                GPIO.output(step, 1)
                time.sleep(0.0000015)
                GPIO.output(step, 0)
                time.sleep(delay_nema)
          if (kroky_krokac1 >= 0 and kroky_krokac2 >= 0):
                if (kroky_krokac1 >= kroky_krokac2):
                      for i in range(kroky_krokac2):
                            for j in range(8):
                                  krok_krokac1(sekvence2[j][0],
sekvence2[j][1], sekvence2[j][2],sekvence2[j][3])
                                  krok_krokac2(sekvence1[j][0],
sekvence1[j][1], sekvence1[j][2],sekvence1[j][3])
```

```
                                    time.sleep(delay_krokace)
                        for i in range(kroky_krokac1 - kroky_krokac2):
                                    for j in range(8):
                                          krok_krokac1(sekvence2[j][0],
sekvence2[j][1], sekvence2[j][2],sekvence2[j][3])
                                    time.sleep(delay_krokace)
                  else:
                        for i in range(kroky_krokac1):
                                    for j in range(8):
                                          krok_krokac1(sekvence2[j][0],
sekvence2[j][1], sekvence2[j][2],sekvence2[j][3])
                                          krok_krokac2(sekvence1[j][0],
sekvence1[j][1], sekvence1[j][2],sekvence1[j][3])
                                    time.sleep(delay_krokace)
                        for i in range(kroky_krokac2 - kroky_krokac1):
                                    for j in range(8):
                                          krok_krokac2(sekvence1[j][0],
sekvence1[j][1], sekvence1[j][2],sekvence1[j][3])
                                    time.sleep(delay_krokace)
            elif(kroky_krokac1 < 0 and kroky_krokac2 < 0):
                  if (abs(kroky_krokac1) >= abs(kroky_krokac2)):
                        for i in range(abs(kroky_krokac2)):
                                    for j in range(8):
                                          krok_krokac1(sekvence1[j][0],
sekvence1[j][1], sekvence1[j][2],sekvence1[j][3])
                                          krok_krokac2(sekvence2[j][0],
sekvence2[j][1], sekvence2[j][2],sekvence2[j][3])
                                    time.sleep(delay_krokace)
                        for i in range(abs(kroky_krokac1) -
abs(kroky_krokac2)):
                                    for j in range(8):
                                          krok_krokac1(sekvence1[j][0],
sekvence1[j][1], sekvence1[j][2],sekvence1[j][3])
                                    time.sleep(delay_krokace)
                  else:
                        for i in range(abs(kroky_krokac1)):
                                    for j in range(8):
                                          krok_krokac1(sekvence1[j][0],
sekvence1[j][1], sekvence1[j][2],sekvence1[j][3])
                                          krok_krokac2(sekvence2[j][0],
sekvence2[j][1], sekvence2[j][2],sekvence2[j][3])
                                    time.sleep(delay_krokace)
                        for i in range(abs(kroky_krokac2) -
abs(kroky_krokac1)):
                                    for j in range(8):
                                          krok_krokac2(sekvence2[j][0],
sekvence2[j][1], sekvence2[j][2],sekvence2[j][3])
                                    time.sleep(delay_krokace)
            elif(kroky_krokac1 >= 0 and kroky_krokac2 < 0):
```

```
            if (abs(kroky_krokac1) >= abs(kroky_krokac2)):
                  for i in range(abs(kroky_krokac2)):
                        for j in range(8):
                              krok_krokac1(sekvence2[j][0],
sekvence2[j][1], sekvence2[j][2],sekvence2[j][3])
                              krok_krokac2(sekvence2[j][0],
sekvence2[j][1], sekvence2[j][2],sekvence2[j][3])
                              time.sleep(delay_krokace)
                  for i in range(abs(kroky_krokac1) -
abs(kroky_krokac2)):
                        for j in range(8):
                              krok_krokac1(sekvence2[j][0],
sekvence2[j][1], sekvence2[j][2],sekvence2[j][3])
                              time.sleep(delay_krokace)
            else:
                  for i in range(abs(kroky_krokac1)):
                        for j in range(8):
                              krok_krokac1(sekvence2[j][0],
sekvence2[j][1], sekvence2[j][2],sekvence2[j][3])
                              krok_krokac2(sekvence2[j][0],
sekvence2[j][1], sekvence2[j][2],sekvence2[j][3])
                              time.sleep(delay_krokace)
                  for i in range(abs(kroky_krokac2) -
abs(kroky_krokac1)):
                        for j in range(8):
                              krok_krokac2(sekvence2[j][0],
sekvence2[j][1], sekvence2[j][2],sekvence2[j][3])
                              time.sleep(delay_krokace)
      else:
            if (abs(kroky_krokac1) >= abs(kroky_krokac2)):
                  for i in range(abs(kroky_krokac2)):
                        for j in range(8):
                              krok_krokac1(sekvence1[j][0],
sekvence1[j][1], sekvence1[j][2],sekvence1[j][3])
                              krok_krokac2(sekvence1[j][0],
sekvence1[j][1], sekvence1[j][2],sekvence1[j][3])
                              time.sleep(delay_krokace)
                  for i in range(abs(kroky_krokac1) -
abs(kroky_krokac2)):
                        for j in range(8):
                              krok_krokac1(sekvence1[j][0],
sekvence1[j][1], sekvence1[j][2],sekvence1[j][3])
                              time.sleep(delay_krokace)
            else:
                  for i in range(abs(kroky_krokac1)):
                        for j in range(8):
                              krok_krokac1(sekvence1[j][0],
sekvence1[j][1], sekvence1[j][2],sekvence1[j][3])
```

```
                                krok_krokac2(sekvence1[j][0],
sekvence1[j][1], sekvence1[j][2],sekvence1[j][3])
                                time.sleep(delay_krokace)
                    for i in range(abs(kroky_krokac2) -
abs(kroky_krokac1)):
                          for j in range(8):
                                krok_krokac2(sekvence1[j][0],
sekvence1[j][1], sekvence1[j][2],sekvence1[j][3])
                                time.sleep(delay_krokace)
    else:
          if (kroky_krokac1 >= 0 and kroky_krokac2 >= 0):
               if (kroky_krokac1 >= kroky_krokac2):
                    for i in range(kroky_krokac2):
                          for j in range(8):
                                krok_krokac1(sekvence2[j][0],
sekvence2[j][1], sekvence2[j][2],sekvence2[j][3])
                                krok_krokac2(sekvence1[j][0],
sekvence1[j][1], sekvence1[j][2],sekvence1[j][3])
                                time.sleep(delay_krokace)
                    for i in range(kroky_krokac1 - kroky_krokac2):
                          for j in range(8):
                                krok_krokac1(sekvence2[j][0],
sekvence2[j][1], sekvence2[j][2],sekvence2[j][3])
                                time.sleep(delay_krokace)
               else:
                    for i in range(kroky_krokac1):
                          for j in range(8):
                                krok_krokac1(sekvence2[j][0],
sekvence2[j][1], sekvence2[j][2],sekvence2[j][3])
                                krok_krokac2(sekvence1[j][0],
sekvence1[j][1], sekvence1[j][2],sekvence1[j][3])
                                time.sleep(delay_krokace)
                    for i in range(kroky_krokac2 - kroky_krokac1):
                          for j in range(8):
                                krok_krokac2(sekvence1[j][0],
sekvence1[j][1], sekvence1[j][2],sekvence1[j][3])
                                time.sleep(delay_krokace)
          elif(kroky_krokac1 < 0 and kroky_krokac2 < 0):
               if (abs(kroky_krokac1) >= abs(kroky_krokac2)):
                    for i in range(abs(kroky_krokac2)):
                          for j in range(8):
                                krok_krokac1(sekvence1[j][0],
sekvence1[j][1], sekvence1[j][2],sekvence1[j][3])
                                krok_krokac2(sekvence2[j][0],
sekvence2[j][1], sekvence2[j][2],sekvence2[j][3])
                                time.sleep(delay_krokace)
                    for i in range(abs(kroky_krokac1) -
abs(kroky_krokac2)):
                          for j in range(8):
```

```
                                krok_krokac1(sekvence1[j][0],
sekvence1[j][1], sekvence1[j][2],sekvence1[j][3])
                                time.sleep(delay_krokace)
                else:
                        for i in range(abs(kroky_krokac1)):
                                for j in range(8):
                                krok_krokac1(sekvence1[j][0],
sekvence1[j][1], sekvence1[j][2],sekvence1[j][3])
                                krok_krokac2(sekvence2[j][0],
sekvence2[j][1], sekvence2[j][2],sekvence2[j][3])
                                time.sleep(delay_krokace)
                        for i in range(abs(kroky_krokac2) -
abs(kroky_krokac1)):
                                for j in range(8):
                                krok_krokac2(sekvence2[j][0],
sekvence2[j][1], sekvence2[j][2],sekvence2[j][3])
                                time.sleep(delay_krokace)
        elif(kroky_krokac1 >= 0 and kroky_krokac2 < 0):
                if (abs(kroky_krokac1) >= abs(kroky_krokac2)):
                        for i in range(abs(kroky_krokac2)):
                                for j in range(8):
                                krok_krokac1(sekvence2[j][0],
sekvence2[j][1], sekvence2[j][2],sekvence2[j][3])
                                krok_krokac2(sekvence2[j][0],
sekvence2[j][1], sekvence2[j][2],sekvence2[j][3])
                                time.sleep(delay_krokace)
                        for i in range(abs(kroky_krokac1) -
abs(kroky_krokac2)):
                                for j in range(8):
                                krok_krokac1(sekvence2[j][0],
sekvence2[j][1], sekvence2[j][2],sekvence2[j][3])
                                time.sleep(delay_krokace)
                else:
                        for i in range(abs(kroky_krokac1)):
                                for j in range(8):
                                krok_krokac1(sekvence2[j][0],
sekvence2[j][1], sekvence2[j][2],sekvence2[j][3])
                                krok_krokac2(sekvence2[j][0],
sekvence2[j][1], sekvence2[j][2],sekvence2[j][3])
                                time.sleep(delay_krokace)
                        for i in range(abs(kroky_krokac2) -
abs(kroky_krokac1)):
                                for j in range(8):
                                krok_krokac2(sekvence2[j][0],
sekvence2[j][1], sekvence2[j][2],sekvence2[j][3])
                                time.sleep(delay_krokace)
        else:
                if (abs(kroky_krokac1) >= abs(kroky_krokac2)):
                        for i in range(abs(kroky_krokac2)):
```

```
                        for j in range(8):
                              krok_krokac1(sekvence1[j][0],
sekvence1[j][1], sekvence1[j][2],sekvence1[j][3])
                              krok_krokac2(sekvence1[j][0],
sekvence1[j][1], sekvence1[j][2],sekvence1[j][3])
                              time.sleep(delay_krokace)
                  for i in range(abs(kroky_krokac1) -
abs(kroky_krokac2)):
                        for j in range(8):
                              krok_krokac1(sekvence1[j][0],
sekvence1[j][1], sekvence1[j][2],sekvence1[j][3])
                              time.sleep(delay_krokace)
            else:
                  for i in range(abs(kroky_krokac1)):
                        for j in range(8):
                              krok_krokac1(sekvence1[j][0],
sekvence1[j][1], sekvence1[j][2],sekvence1[j][3])
                              krok_krokac2(sekvence1[j][0],
sekvence1[j][1], sekvence1[j][2],sekvence1[j][3])
                              time.sleep(delay_krokace)
                  for i in range(abs(kroky_krokac2) -
abs(kroky_krokac1)):
                        for j in range(8):
                              krok_krokac2(sekvence1[j][0],
sekvence1[j][1], sekvence1[j][2],sekvence1[j][3])
                              time.sleep(delay_krokace)
      GPIO.output(direction, 0)
      time.sleep(0.005)
      for i in range (abs(kroky_nema)):
            GPIO.output(step, 1)
            time.sleep(0.0000015)
            GPIO.output(step, 0)
            time.sleep(delay_nema)
#mezkikrok zapínající/vypínající motor SX17
def najed_do_pozice(req):
    GPIO.output(sleep, 1)
    time.sleep(0.005)
    najeti_do_pozice(req.nema, req.krokac1, req.krokac2)
    GPIO.output(sleep, 0)
    return polohovaniResponse('Hotovo')
#funkce definující službu v ROS
def polohovani_service():
    rospy.init_node('pohyb')
    s = rospy.Service('polohovani', polohovani, najed_do_pozice)
    rospy.spin()

if __name__ == "__main__":
    nastav_GPIO()
    try:
```

```
        polohovani_service()
    except KeyboardInterrupt:
        GPIO.cleanup()
        exit()
```

homing.py

```
#!/usr/bin/env python

import rospy
import RPi.GPIO as GPIO
import time
from r_ruka.srv import *

#deklarace hodnot pinů motorů a koncových spínačů a časových
#rozestupů mezi kroky motorů
direction = 26
step = 19
sleep = 13
endstop_nema = 7
endstop1 = 25
endstop2 = 8
krokac1 = [4, 17, 27, 22]
krokac2 = [12, 16, 20, 21]
delay_nema = 0.00101
delay_krokace = 0.002
#sekvence pro krokování motoru druhého ramena směrem ke koncovým
#spínačům
sekvence1 = range(0, 8)
sekvence1[0] = [1,0,0,1]
sekvence1[1] = [0,0,0,1]
sekvence1[2] = [0,0,1,1]
sekvence1[3] = [0,0,1,0]
sekvence1[4] = [0,1,1,0]
sekvence1[5] = [0,1,0,0]
sekvence1[6] = [1,1,0,0]
sekvence1[7] = [1,0,0,0]
#sekvence pro krokování motoru třetího ramena směrem ke koncovým
#spínačům (obrácená sekvence1)
sekvence2 = range(0, 8)
sekvence2[0] = [1,0,0,0]
sekvence2[1] = [1,1,0,0]
sekvence2[2] = [0,1,0,0]
sekvence2[3] = [0,1,1,0]
sekvence2[4] = [0,0,1,0]
sekvence2[5] = [0,0,1,1]
sekvence2[6] = [0,0,0,1]
sekvence2[7] = [1,0,0,1]
#funkce pro nastaveni rozhraní GPIO
```

```
def nastav_GPIO():
    GPIO.setmode(GPIO.BCM)
    GPIO.setwarnings(False)
    GPIO.setup(direction, GPIO.OUT)
    GPIO.output(direction, 1)
    GPIO.setup(step, GPIO.OUT)
    GPIO.output(step,0)
    GPIO.setup(sleep, GPIO.OUT)
    GPIO.output(sleep, 0)
    for pin in range(4):
          GPIO.setup(krokac1[pin],GPIO.OUT)
          GPIO.output(krokac1[pin], 0)
          GPIO.setup(krokac2[pin],GPIO.OUT)
          GPIO.output(krokac2[pin], 0)
#funkce nastavující hodnoty na pinech pro provedení kroku motoru
#druhého ramena
def krok_krokac1(p1, p2, p3, p4):
    GPIO.output(krokac1[0], p1)
    GPIO.output(krokac1[1], p2)
    GPIO.output(krokac1[2], p3)
    GPIO.output(krokac1[3], p4)
#funkce nastavující hodnoty na pinech pro provedení kroku motoru
#třetího ramena
def krok_krokac2(p1, p2, p3, p4):
    GPIO.output(krokac2[0], p1)
    GPIO.output(krokac2[1], p2)
    GPIO.output(krokac2[2], p3)
    GPIO.output(krokac2[3], p4)
#funkce pro najetí motoru do základní pozice ze známé pozice,
#ramena postupně od prvního na dorazy
def homing_z_pozice():
    GPIO.output(sleep, 1)
    time.sleep(0.0015)
    while (GPIO.input(endstop_nema) != 0):
          GPIO.output(step, 1)
          time.sleep(0.0000015)
          GPIO.output(step, 0)
          time.sleep(delay_nema)
    GPIO.output(sleep, 0)
    p = 8
    while (GPIO.input(endstop1) != 0):
          krok_krokac1(sekvence1[p % 8][0], sekvence1[p % 8][1],
sekvence1[p % 8][2], sekvence1[p % 8][3])
          time.sleep(delay_krokace)
          p += 1
    p = 8
    while (GPIO.input(endstop2) != 0):
          krok_krokac2(sekvence2[p % 8][0], sekvence2[p % 8][1],
sekvence2[p % 8][2], sekvence2[p % 8][3])
```

```
            time.sleep(delay_krokace)
            p += 1
#funkce pro najetí motoru do základní pozice z neznámé pozice
def homing_random():
    if ( (GPIO.input(endstop_nema) == 0) and (GPIO.input(endstop1)
== 0) and (GPIO.input(endstop2) == 0) ):
            pass
    elif ( (GPIO.input(endstop_nema) == 0) and (GPIO.input(endstop1)
== 0) ):
            p = 8
            while (GPIO.input(endstop2) != 0):
                  krok_krokac2(sekvence2[p % 8][0], sekvence2[p %
8][1], sekvence2[p % 8][2], sekvence2[p % 8][3])
                  time.sleep(delay_krokace)
                  p += 1
    elif (GPIO.input(endstop_nema) == 0):
            for i in range(20):
                  for j in range(8):
                        krok_krokac2(sekvence1[j][0], sekvence1[j][1],
sekvence1[j][2],sekvence1[j][3])
                        time.sleep(delay_krokace)
            p = 8
            while (GPIO.input(endstop1) != 0):
                  krok_krokac1(sekvence1[p % 8][0], sekvence1[p %
8][1], sekvence1[p % 8][2], sekvence1[p % 8][3])
                  time.sleep(delay_krokace)
                  p += 1
            p = 8
            while (GPIO.input(endstop2) != 0):
                  krok_krokac2(sekvence2[p % 8][0], sekvence2[p %
8][1], sekvence2[p % 8][2], sekvence2[p % 8][3])
                  time.sleep(delay_krokace)
                  p += 1
    else:
            for i in range(30):
                  for j in range(8):
                        krok_krokac1(sekvence2[j][0], sekvence2[j][1],
sekvence2[j][2],sekvence2[j][3])
                        krok_krokac2(sekvence1[j][0], sekvence1[j][1],
sekvence1[j][2],sekvence1[j][3])
                        time.sleep(delay_krokace)
            GPIO.output(sleep, 1)
            time.sleep(0.0015)
            while (GPIO.input(endstop_nema) != 0):
                  GPIO.output(step, 1)
                  time.sleep(0.0000015)
                  GPIO.output(step, 0)
                  time.sleep(delay_nema)
            GPIO.output(sleep, 0)
```

```
            p = 8
            while (GPIO.input(endstop1) != 0):
                  krok_krokac1(sekvence1[p % 8][0], sekvence1[p %
8][1], sekvence1[p % 8][2], sekvence1[p % 8][3])
                  time.sleep(delay_krokace)
                  p += 1
            p = 8
            while (GPIO.input(endstop2) != 0):
                  krok_krokac2(sekvence2[p % 8][0], sekvence2[p %
8][1], sekvence2[p % 8][2], sekvence2[p % 8][3])
                  time.sleep(delay_krokace)
                  p += 1
#funkce rozhodující o průběhu najíždění
def homing(req):
    GPIO.output(direction, 1)
    if req.info == 'pozice':
          homing_z_pozice()
    else:
          homing_random()
    return homeResponse('Hotovo')
#funkce definující službu v ROS
def service_homing():
    rospy.init_node('homing')
    s = rospy.Service('homovani', home, homing)
    rospy.spin()

if __name__ == "__main__":
    nastav_GPIO()
    try:
          service_homing()
    except KeyboardInterrupt:
          GPIO.cleanup()
```

kontrola_pozice.py

```
#!/usr/bin/env python

import rospy
import RPi.GPIO as GPIO
from r_ruka.srv import *

infra_cidlo = 5
#funkce pro nastaveni rozhraní GPIO
def nastav_GPIO():
    GPIO.setmode(GPIO.BCM)
    GPIO.setwarnings(False)
    GPIO.setup(infra_cidlo, GPIO.IN)
#funkce vyhodnocující hodnotu z čidla
```

```
def zjisti(req):
    if (GPIO.input(infra_cidlo) == 0):
          return objektResponse('obsazeno')
    else:
          return objektResponse('prazdno')
#funkce definující službu v ROS
def zjistovani_service():
    rospy.init_node('kontrola_pozice')
    s = rospy.Service('zjisteni', objekt, zjisti)
    rospy.spin()

if __name__ == '__main__' :
    nastav_GPIO()
    try:
          zjistovani_service()
    except KeyboardInterrupt:
          GPIO.cleanup()
          exit()
```

kontrola_uchopeni.py

```
#!/usr/bin/env python

import rospy
import RPi.GPIO as GPIO
from r_ruka.srv import *

infra_cidlo = 6
#funkce pro nastaveni rozhraní GPIO
def nastav_GPIO():
    GPIO.setmode(GPIO.BCM)
    GPIO.setwarnings(False)
    GPIO.setup(infra_cidlo, GPIO.IN)
#funkce vyhodnocující hodnotu z čidla
def zjisti(req):
    if (GPIO.input(infra_cidlo) == 0):
          return objektResponse('uchopen')
    else:
          return objektResponse('prazdno')
#funkce definující službu v ROS
def zjistovani_service():
    rospy.init_node('kontrola_uchopeni')
    s = rospy.Service('kontrola_uchopeni', objekt, zjisti)
    rospy.spin()

if __name__ == '__main__' :
    nastav_GPIO()
    try:
          zjistovani_service()
```

```
    except KeyboardInterrupt:
        GPIO.cleanup()
        exit()
```

kontrola_vzdalenosti.py

```
#!/usr/bin/env python

import rospy
import RPi.GPIO as GPIO
import time
from r_ruka.srv import *

ultrazvuk_echo = 23
ultrazvuk_trig = 24
#funkce pro nastaveni rozhraní GPIO
def nastav_GPIO():
    GPIO.setmode(GPIO.BCM)
    GPIO.setwarnings(False)
    GPIO.setup(ultrazvuk_trig, GPIO.OUT)
    GPIO.setup(ultrazvuk_echo, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
    GPIO.output(ultrazvuk_trig, GPIO.LOW)
#funkce pro porovnání očekávané vzdálenosti s naměřenou
def zmer_vzdalenost(req):
#měření vzdálenosti
    soucet = 0.0
    for i in range(3):
        GPIO.output(ultrazvuk_trig, 1)
        time.sleep(0.00001)
        GPIO.output(ultrazvuk_trig, 0)

        while GPIO.input(ultrazvuk_echo)==0:
            pulse_start_time = time.time()
        while GPIO.input(ultrazvuk_echo)==1:
            pulse_end_time = time.time()
#výpočet vzdálenosti z poloviny délky signálu vysoké hodnoty a
#rychlosti zvuku
        pulse_duration = pulse_end_time - pulse_start_time
        vzdalenost = pulse_duration * 171500
        soucet += vzdalenost
#porovnání vzdáleností
    if ( (req.ocekavana - 3) > (soucet / 3) > (req.ocekavana + 3) ):
        return vzdalenostResponse('NOK')
    else:
        return vzdalenostResponse('OK')
#funkce definující službu v ROS
```

```
def vzdalenost_service():
    rospy.init_node('kontrola_vzdalenosti')
    s = rospy.Service('porovnani_vzdalenosti', vzdalenost,
zmer_vzdalenost)
    rospy.spin()

if __name__ == '__main__':
    nastav_GPIO()
    try:
          vzdalenost_service()
    except KeyboardInterrupt:
          GPIO.cleanup()
          exit()
```

tlacitko.py

```
#!/usr/bin/env python

import rospy
import RPi.GPIO as GPIO
from r_ruka.srv import *

tlacitko = 15
#funkce pro nastaveni rozhraní GPIO, pull_up_down-přidání rezistoru
#pro stabilizaci vstupního napětí
def nastav_GPIO():
    GPIO.setmode(GPIO.BCM)
    GPIO.setwarnings(False)
    GPIO.setup(tlacitko, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
#funkce pro čekání na stisk tlačítka
def pockej_na_stisk(req):
    while GPIO.input(tlacitko) == False:
          pass
    return stisknutiResponse('Stisknuto')
#funkcece obsahující definici služby v ROS
def tlacitko_service():
    rospy.init_node('tlacitko')
    s = rospy.Service('stisk', stisknuti, pockej_na_stisk)
    rospy.spin()

if __name__ == '__main__' :
    nastav_GPIO()
    try:
          tlacitko_service()
    except KeyboardInterrupt:
          GPIO.cleanup()
          exit()
```

C Ostatní přílohy

Přiložené CD obsahuje elektronickou verzi práce ve formátu PDF. Dále obsahuje zdrojové soubory jednotlivých uzlů, uložených v adresáři Roboticka_ruka.