České vysoké učení technické
Fakulta strojní
Ústav mechaniky, biomechaniky a mechatroniky

Robot ovládaný gesty

Diplomová práce

Bc. David Jakubec

Navazující magisterský program: Průmysl 4.0

Vedoucí: Ing. Jan Zavřel, Ph.D.

Anotační list

Jméno autora:	**David Jakubec**
Název diplomové práce:	**Robot ovládaný gesty**
Anglický název:	**Manipulation robot controlled by gestures**
Akademický rok:	2019/2020
Studijní obor:	Průmysl 4.0
Ústav, odbor:	Ústav mechaniky, biomechaniky a mechatroniky Odbor mechaniky a mechatroniky
Vedoucí bakalářské práce:	Ing. Jan Zavřel, Ph.D
Bibliografické údaje:	77 stran 53 obrázků příloha – CD
Klíčová slova:	Mechatronický manipulátor, kinematika, neuronové sítě, počítačové vidění
Key words:	Mechatronical manipulator, kinematics, neural networks, computer vision

Abstrakt: Cílem této práce je navrhnout mechanický manipulátor ovládaný gesty s vhodnou kinematickou strukturou a řešením pro ovládání. Pro rozpoznání gest je využito strojového vidění a neuronových sítí. V práci je shrnuta rešerše hardwarových vývojových desek, seznámení se s ovládáním prostorových mechanismů a kalibrace.

Abstract: The aim of this thesis is to design a mechanical manipulator controlled by gestures with suitable kinematic structure. For gesture recognition is used computer vision and neural networks. This work features research of hardware components such as development board, familiarization with control of spatial mechanisms and calibration.

Prohlášení

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

v Praze, dne 12.8.2020

.....

Bc. David Jakubec

Obsah

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod

V rámci každé průmyslové revoluce bylo z průmyslu propuštěno mnoho lidí z důvodu zavádění strojů. Míra automatizace se zvyšuje a potřeba lidí v průmyslu klesá díky pokroku a neustálému vývoji umělé inteligence i mechanických systémů. Iniciativa Průmysl 4.0 klade větší nároky na automatizaci, která zvyšuje efektivitu výroby a podnikům tímto přináší větší zisky. Ale všechny práce nejdou plně zautomatizovat a tudíž je člověk nedílnou součástí výrobního systému.

Hlavním využitím lidského faktoru v průmyslu je syntéza výkonnosti mozku a zručnosti rukou. Problém využití člověka na pracovišti spočívá v tom, že cítí únavu a stres. Při špatné hygieně práce může být kritickým faktorem pro nesplnění předepsaných výrobních kvóta a následkem toho může být velká zmetkovitost a občas i úraz. Kamerovým systémem lze i zabránit například detekování srážek pracovníka a nepozorného operátora manipulačního vozíku, je možné kontrolovat překročení hygienických předpisů, špatná manipulace s výrobky a jiné. V některých případech je i dlouhodobý pobyt v pracovním prostředí pro člověka nebezpečný a je nezbytný ochranný oblek. Kupříkladu u výkonu práce v prostředí se zvýšenou radiací, ve slévárenských místnostech s vysokými teplotami či při práci s kovovým 3D tiskem.

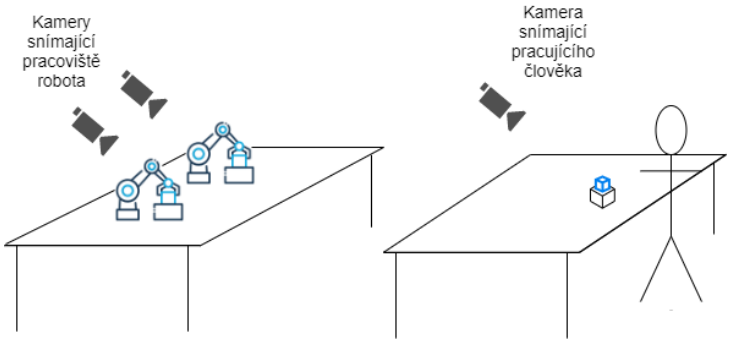
Nynější mechatronické systémy a umělá inteligence jsou již na vyspělé úrovni a je relevantní si pokládat otázku, zda stroje opravdu lidskou pracovní sílu nahradí. Výše zmíněným kamerovým systémem lze v průběhu několika let získat dostatečné množství dat pro analýzu pohybu lidí i jejich končetin vykonávajících práci v průmyslu. Při

syntéze s antropomorfním mechatronickým systémem společně s deep learningem je teoreticky možné naučit roboty odvádět univerzální práci místo lidí. Jako další možností je nasazení robotů a jejich ovládání pomocí rozšířené reality. Kvalifikovaný člověk ovládající robota, přes vysokorychlostní síť s nízkým zpožděním signálu, bude moci vykonat práci v průmyslu vzdáleně. Z vykonávané činnosti se může umělá inteligence učit. Je možné v průběhu let nahradit lidi roboty?

Uvádím možné motivace řešení pro ovládání robota, kterými jsou paralelní práce a teleoperace robota.

Paralelní práce

Při využití paralelní práce lze sledovat pracovníka kamerou a vyhodnocovat pozici jeho ruky společně s rozpoznáním všech předmětů, které má pracovník na stole. Pozice všech předmětů i ruky se může zaznamenávat v čase. V případě, že snímaný předmět a ruka bude mít velký rozdíl snímaných souřadnic, pak se podle korelace pohybů může rozpoznat, který objekt byl sebrán. Při další změně rozdílů poloh je zřejmé, že člověk s předmětem již nemanipuluje.

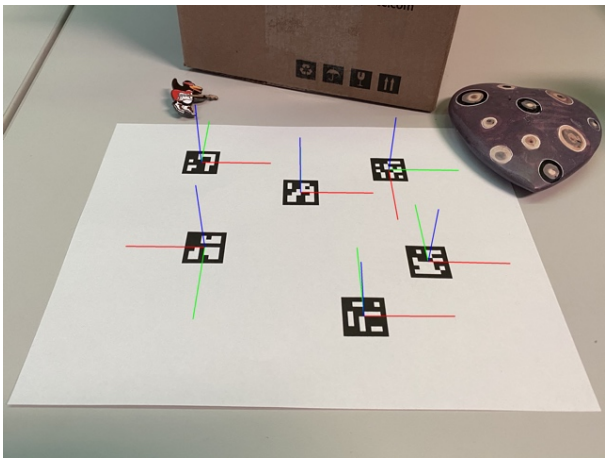


Obrázek 1.1 – Paralelní práce

Informace, jaký předmět byl uchopen a kam byl položen s ohledem na natočení jsou poslány, do robota. Kamery snímající robotické pracoviště vyhledají kde se předmět nachází, vyhodnotí jeho souřadnice a robot vezme objekt, který položí na stejné místo jako pracovník.

Pro zajištění souřadnicového systému každého robota i pracovníka je možné použít například štítkovací systém knihovny Aruco pro jazyk C++ i Python [1]. Tato knihovna obsahuje rozpoznávání dle štítků, pomocí kterých lze definovat souřadnicový systém.

Po definování souřadnicového systému je nutné robota zkalibrovat s ohledem na daný souřadnicový systém.



Obrázek 1.2 – Použití QR kódu k definování globálního souřadnicového systému [1]

Teleoperace robota

Počítačovým viděním je analyzována postura pracovníka, ze které se odečítají úhly natočení v kloubech ruky. Pokud je typ kamery RGB-D, tak se z analyzovaného kloubu, který zachytí na snímku v pozici v pixelech X, Y a odečte vzdálenost. Přepočtením je možné získat polohu bodů X, Y, Z v kartézských souřadnicích a nikoliv hodnoty X, Y v pixelech a Z v metrech. Použitím například balíčku OpenPose [2], je možné získat virtuální kostru člověka, podle které se bude definovat pohyb robota.

Tímto způsobem získám pohyb robotického manipulátoru k ruce jedna ku jedné. Motivace k použití takového robota vzniká v podmínkách, které jsou pro člověka hůře dostupné, v prostředí, ve kterém člověk nevydrží dlouho či je životu nebezpečné. Využití se najde také při manipulaci s velmi těžkými předměty, které vyžadují zvláštní zacházení. Při použití tlakových senzorů v koncovém efektoru robota je možné s haptickou rukavicí simulovat zpětnou vazbu a člověk neztratí hmat úplně, což usnadní ovládání robota. Robot disponující kamerou se schopností rotace, a přenášení obrazových informací, který v tu chvíli pracovník uvidí očima robota, může zaručit nahrazení ovládání. Zobrazování může probíhat pomocí brýlích pro virtuální realitu, nebo pouze na obrazovce. Přesnost robota bude ovšem záviset na přesnosti odměřování klíčových bodů a přesnosti robota.

Kapitola 2

Cíle práce

Pro vypracování diplomové práce jsou stanoveny tyto cíle:

1. Provést rešerši manipulačních robotů
2. Seznámit se s řešením kinematiky prostorových mechanismů
3. Navrhnout strukturu robota pro manipulaci
4. Navrhnout konstrukci a hardwarové řešení pro řízení robota

Kapitola 3

Rešerše

3.1 Manipulační zařízení [3]

Robotický manipulátor se dělí do několika skupin, podle obrázku 3.1. Jednoúčelové manipulátory jsou speciální skupinou manipulačních zařízení, které nemají široké spektrum možností pro manipulování. Jednoúčelový manipulátor je charakterizován tím, že podavač může být součástí stroje a manipuluje s malou sadou objektů různého tvaru dle potřeby uživatele, se kterými se univerzální manipulátory nemusí umět poradit.

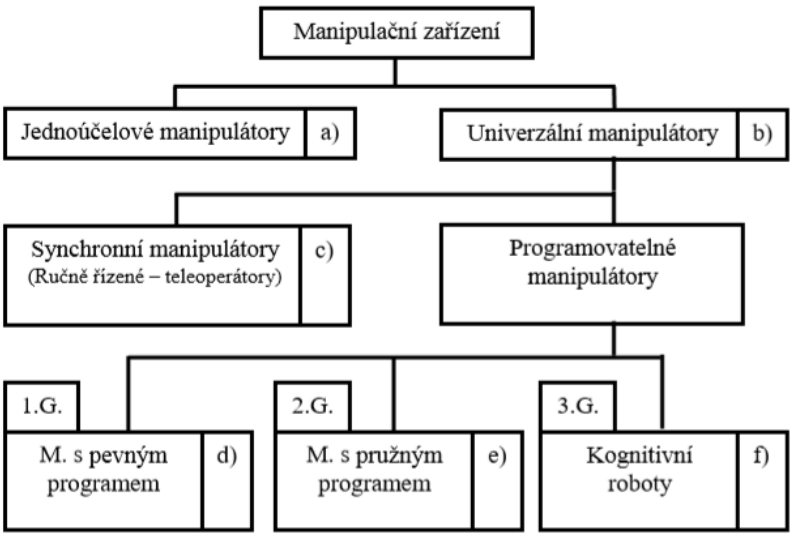
Univerzální manipulátor je opak jednoúčelového. Má větší rozsah manipulace a je nezávislý na typu manipulované součásti a stroje. Je charakterizován mechanickými parametry jako počet stupňů volnosti, přesností polohování, rozsahem dílčích pohybů a maximálním zatěžováním. Příkladem univerzálního manipulátoru jsou například klasické robotické manipulátory od firmy Kuka, ABB či Fanuc. Tyto manipulátory se poté dělí na synchronní a programovatelné.

Synchronní manipulátor operuje člověk, který ovládá robota pomocí joysticku, nebo jiného zařízení, kde se převádí elektrický signál do řídícího členu robota, který následně vykoná pohyb s určitým zesílením. Výsledkem je to, že na rozdíl od člověka, dobře zkonstruovaný a zkalibrovaný robot má větší přesnost a jsou redukovány vibrace (např. třes rukou je vyloučen). Proto takové manipulátory jsou dobré pro drobné práce či chirurgické zákroky. Další možností je využití kamerového systému k provádění

synchronních operací, čímž se tato práce zabývá.

Dále existuje třída programovatelných manipulátorů. Ty se dělí na manipulátory s programem pevným a pružným a na kognitivní roboty. Manipulátor s pevným programem může být ovládán například vačkovou hřídelí. Jelikož by bylo potřeba vyrobit novou vačkovou hřídel, aby mohl být program změněn, je změna chování velice náročná a nákladná. Oproti tomu jeho nástupce, manipulátor s pružným programem, je řízen elektronicky a je programovatelný. z dat čtených ze senzorů a čidel se můžou programy nastavovat podle aktuálních hodnot a ovládání je tímto parametrizované.

Poslední třída kognitivní roboty, je třetí z generace programovatelných manipulátorů. Kognitivnost spočívá ve „vnímání“ okolí a rozpoznání objektů za pomoci strojového vidění a neuronových sítí. Dle rozpoznání se robot může zachovat dynamicky, učit se nové věci, adaptovat, hledat optimální variantu, řídit paralelní procesy nebo predikovat za pomoci algoritmů umělé inteligence. Druhá stránka věci je ta, že výsledky algoritmů mohou být mlhavé či nepřesné a robot se nemusí přizpůsobit správně a je nutno robota doučit nebo přeučit, aby se přesnost byla co nejlepší a chyba se snížila na minimum.



Obrázek 3.1 – Rozdělení manipulačních zařízení dle provedení a stupně řízení

3.2 Řešení kinematických struktur

S prostorovými mechanismy se setkáváme ve všech oborech s různorodými stroji a zařízeními. V každém případě takové stroje potřebují způsob řízení, aby plnily svůj úkol.

Základem pro vypracování metod kinematické analýzy, je kinematika prostorového bodu a tělesa v maticové formulaci. Ve valné většině případů se souřadnicové systémy zapisují v pravoúhlém pravotočivém systému. Ke koncovému bodu v kinematice se mohu dostat základními transformacemi, translací a rotací [4].

V případě obecného prostoru souřadnicového systému A a souřadnicového systému B, je obecný vztah transformační matice mezi nimi udán vztahem 3.1, kde S_{AB} je matice směrových kosinů v prostoru A - natočení souřadného systému v prostoru. u_{AB} je průvodič z bodu B v prostoru A [4].

$$T_{AB} = \begin{bmatrix} \mathbf{S}_{AB} & \mathbf{u}_{AB} \\ 0 & 1 \end{bmatrix} \tag{3.1}$$

Matice směrových kosinů jsou popsány maticemi (3.2), (3.3), (3.4), kde podle indexu značí rotaci podle příslušné osy - φ_x se točí kolem osy x, φ_y kolem osy y, φ_z kolem osy z. Translace je popsaná maticemi (3.5), (3.6), (3.7), kde x, y, z jsou velikosti posuvu v osách lokálního souřadnicového systému [4].

$$T_{\varphi_X}(\varphi_a) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos(\varphi_a) & -sin(\varphi_a) & 0 \\ 0 & sin(\varphi_a) & cos(\varphi_a) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.2}$$

$$T_{\varphi_Y}(\varphi_b) = \begin{bmatrix} cos(\varphi_b) & 0 & sin(\varphi_b) & 0 \\ 0 & 1 & 0 & 0 \\ -sin(\varphi_b) & 0 & cos(\varphi_b) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.3}$$

$$T_{\varphi_Z}(\varphi_c) = \begin{bmatrix} cos(\varphi_c) & 0 & -sin(\varphi_c) & 0 \\ sin(\varphi_c) & 1 & cos(\varphi_c) & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.4}$$

$$T_x(X) = \begin{bmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.5}$$

$$T_y(Y) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.6}$$

$$T_z(Z) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.7}$$

3.2.1 Přímá kinematika

Transformačními maticemi lze popsat kinematickou strukturu až ke koncovému efektoru podle natočení v kloubech či posunutích ve vedeních nebo lineárních motorech. Popis je uskutečněný vynásobením matic tak. Časté je použití Denavit-Hartenberg notace, která je také uváděná pouze použitím iniciálu DH notace. Ta určuje vztah mezi tělesem $i-1$ a tělesem i.

Tato notace se používá hlavně v robotice, skládá se ze čtyř transformačních matic, jejichž složení je popsáno rovnicí (3.8), je konvencí a značí se jako A. DH notace .Jmenovitě jsou tyto parametry vzdálenost spojnice (a_i), natočení spojnice (α_i), ofset spojnice (d_i) a natočení v kloubu. [5]

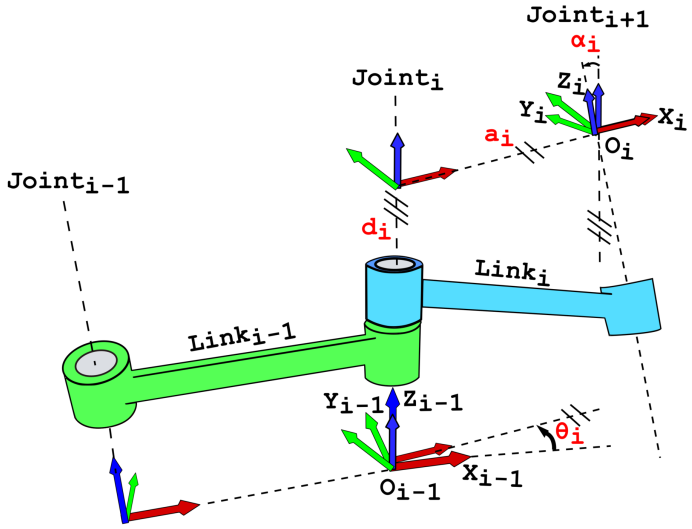
$$A_i = T_{\varphi_z}(\theta_i)T_z(d_i)T_x(a_i)T_{\varphi_x}(\alpha_i) \tag{3.8}$$

Na obrázku 3.2, je příklad vazby. Mezi kloubem i a i+1 je natočení θ, které je z pohonu robota, ofset d pro srovnání s následujícím souřadnicovým systémem, kde po normále

Kloub	θ_i	d_i	a_i	α_i
i-1	θ_{i-1}	0	a_{i-1}	0
i	θ_i	0	a_i	0

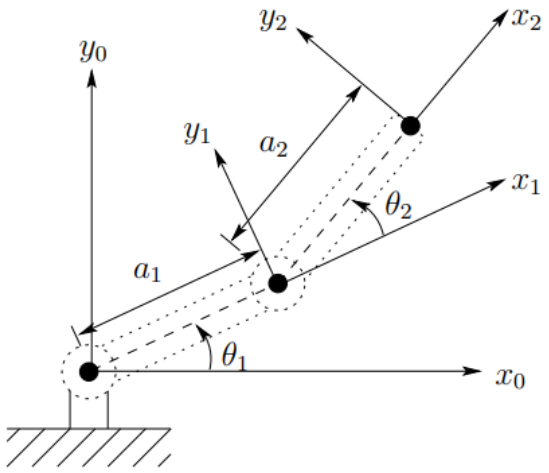
Tabulka 3.1 – Tabulkový zápis DH notace podle obrázku 3.3

k ose transformuji ve vzdálenosti a a jako poslední krok je natočení o úhel α z původního systému do nového.



Obrázek 3.2 – Ukázka dvou prostorových vazeb DH popisu [6]

Pro jednoduchost ilustrace jsem uvedl planárního robota (obr. 3.3), kde jeho DH parametry lze zapsat do přehledné tabulky 3.1, kde všechna d_i a α jsou nulové z toho důvodu, že robot není prostorový.



Obrázek 3.3 – Planární robot

3.2.2 Inverzní kinematika

Inverzní kinematika je řešení úlohy pro kinematickou strukturu, pro kterou jsou zadány souřadnice, do kterého se má koncový efektor robota pohnout. Následně jsou vypočteny úhly natočení či velikost posunutí v pohonech, aby se koncový efektor robota dostal do požadovaného bodu. Přístupy k inverzní kinematice jsou dva - první je analytický a druhý je numerický. U analytického řešení je rovnice $\vec{P_D} = f(\vec{q})$, kde $\vec{P_D}$ je požadovaný bod a $\vec{q}$ jsou hodnoty poloh v motorech, je matematicky vyjádřeno na uzavřený tvar skrze symbolické manipulace. U numerického přístupu je dán počáteční odhad vektor $\vec{q_0}$ a je spočtena sekvence konfigurací $\vec{q_0}$, $\vec{q_1}$, $\vec{q_2}$, ... do té doby, než hodnota chyby $f(\vec{q} - \vec{P_D})$ se bude blížit nule [7].

Výhody analytické metody jsou ty, že se mohou spočítat všechna řešení a určit, zda vůbec řešení existuje. Jakmile je spočítán Jakobián, je řešení velmi rychlé na výpočet. Dále se nemusí určovat počáteční odhad $\vec{q_0}$. Nevýhody analytické metody jsou ty, že při větších kinematických strukturách může být výpočet Jakobiánu problematický. Symbolický výpočet jakobiánu v programu MATLAB vrací příliš dlouhé výsledky, jejichž počet znaků přesahuje počet znaků, které se vejdou do příkazové řádky. Při změně nebo použití rozdílných kinematických struktur se musí znovu vypočítat Jakobián. Další nevýhodou je použití pouze u robotů, jejichž počet stupňů volnosti je roven počtu možných rotací a translací [7] [8].

3.2.3 Kalibrace [9], [10]

U Každé vyrobené součásti může tedy nastat jakákoliv odchylka, která se v robotu projeví na odchylkou polohy koncového efektoru. Tyto odchylky je potřeba odhalit, a tím zvýšit přesnost koncového efektoru robota. K tomu slouží kalibrace. Nastavením robota do různých poloh v pracovním prostoru a precizním odměřováním skutečné polohy v koncovém efektoru můžeme zjistit kalibrační parametry.

Základní algoritmus

Kinematická struktura je popsána vazbovými rovnicemi (3.9) mezi koncovým efektorem a systémem pro odměřování. Do vazbových rovnic vstupují parametry **v** - odměřované souřadnice koncového efektoru, **s** - vstupní odměřované souřadnice (aktuátory)

a **d** - fyzické rozměry mechanismu.

$$\mathbf{f(v,\ s,\ d) = 0} \tag{3.9}$$

Prováděná kalibrace je provedena pro polohy j = 1, 2 . . . , n. Vazbové rovnice jsou dosazené do vektoru **F** z rovnice (3.10), kde **V** je matice parametrů $\mathbf{V} = [v_1, v_2, \ldots, v_n]^T$ pro j-tou polohu, **S** je matice parametrů $\mathbf{S} = [s_1, s_2, \ldots, s_n]^T$. Vektor d je stále stejný pro každou měřenou pozici.

$$\mathbf{F(V,\ S,\ d) = 0} \tag{3.10}$$

Skutečné parametry se liší od navrhovaných a zavede nové značení $\overline{\mathbf{d}}$, což reprezentuje zohlednění výrobních odchylek. Pokud počet kalibračních pozic je vyšší než počet kalibračních parametrů, pak rovnice (3.10) se jedná o přeurčený systém nelineárních rovnic, pro které lze vyhodnocovat kalibrační odchylky. Řeší se pomocí modifikované Newtonovy iterační metody, která je odvozená z Taylorova rozvoje vektoru vazbových rovnic (3.10). z Taylorova rozvoje se získala rovnice (3.11).

$$\mathbf{F}(\mathbf{V}, \mathbf{S}, \overline{\mathbf{d}}) + \mathbf{J}_d \delta\mathbf{d} + \cdots = \mathbf{0} \tag{3.11}$$

$\mathbf{J}_d$ je Jacobiho matice parciálních derivací s ohledem na kalibrační parametry $\overline{\mathbf{d}}$, i-tý krok Newtonovy iterační metody je vyjádřen rovnicí (3.12), kde $\delta\mathbf{d}_i$ je vektor opravy kalibračních parametrů.

$$\delta\mathbf{d}_i = (\mathbf{J}_{\mathbf{d_i}}^T * \mathbf{J}_{\mathbf{d_i}})^{-1} \mathbf{J}_{\mathbf{d_i}}^T (-\mathbf{F}(\mathbf{d_i}, \mathbf{S}, \mathbf{V})) \tag{3.12}$$

kde $-\mathbf{F}(\mathbf{d_i}, \mathbf{S}, \mathbf{V})$ je vektor vypočtených kalibračních odchylek s ohledem na kalibrační parametry z předchozího kroku. Nové kalibrační parametry se získají z rovnice (3.13) sečtením stávajících parametrů a opravených kalibračních parametrů.

$$\mathbf{d}_{i+1} = \mathbf{d}_i + \delta\mathbf{d}_i \tag{3.13}$$

Iterace pokračuje, dokud jsou kalibrační parametry. Kalibrační parametry se budou měnit tím rychleji, čím je menší kalibrovatelnost C, která je dána rovnicí (3.14).

$$C = cond(\mathbf{J_d^T J_d}) \tag{3.14}$$

3.3 Hardware

Mezi nejznámější vývojové desky patří Arduino jako mikrořadič a RaspberryPI jako mikropočítač. Další nástupci jsou novější Coral Dev Board od Googlu a Jetson Nano od společnosti NVIDIA, které jsou osazené hardwarem pro využití strojového učení, strojového vidění a umělou inteligencí. Firma Movidius přišla na trh s produktem NeuralCompute stick, jako USB akcelerátor. O něco později přišel Google se svým produktem Coral. Obojí funguje na principu TPU (Tensor Processing Unit). Vývojové desky, které nepodporují ML (Machine Learning) nebo MV (Machine Vision) mohou používat tyto akcelerátory. Možností řešení je tedy opravdu mnoho.
V této sekci popíši nároky na hardware. Zaměřím se na nízkonákladové vývojové desky, kamerové senzory, aktuátory a další komponenty pro sestavení manipulačního zařízení. Komponenty porovnám pod stejnou cenovou kategorií a budu hledat možné řešení. Hlavní rozbor bude zaměřený na vývojové desky. Jelikož chci ve své práci využít neuronové sítě, vyvstává požadavek na dostatečně rychlou inferenci [1] neuronových sítí, proto tento rychlost inference budu používat jako metriku s velkou váhou.

3.3.1 Vývojové desky

Raspberry Pi

Raspberry Pi (RPi) je jedna z nejlevnějších a nejrozšířenějších vývojových desek. Cena nejnovější RPi4 Model B se pohybuje okolo 1500 Kč. Spadá do kategorie počítače, jelikož je možné na ni nainstalovat operační systém. Existuje okolo 40 [11] distribucí založených na Linuxu a několik na Windows. Kromě dvou nejznámějších, výše zmíněných, operačních systémů existuje i několik distribucí, které nejsou žádný z nich [12]. Každá z distribucí přináší určité výhody a nástroje, které uživatel preferuje. z Raspberry

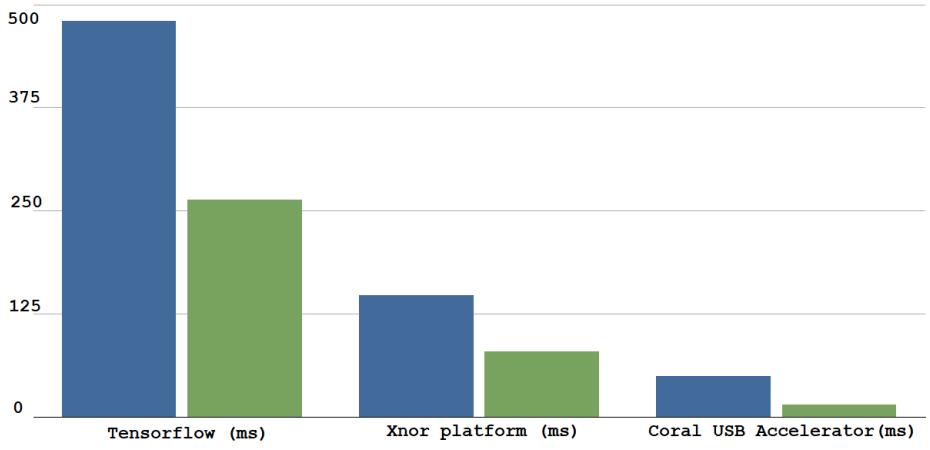
[1] Inferencí nazýváme přeměnu vstupních dat na výstupní u natrénovaných umělých neuronových sítí.

se tímto lehce stane herní konzole, zařízení pro IoT nebo obvyklý stolní počítač [13]. Oficiální verzí je linuxová distribuce Raspbian, která byla vydána poprvé v roce 2012 [12].

Prodává se v několika verzích, podle volby operační paměti LPDDR4 RAM. Dostupné jsou v 1 GB, 2 GB, 4 GB verzi. Obsahuje 28 nm procesor ARM Cortex-A72 s taktovací frekvencí 1,5 GHz a čtyřmi jádry. Grafický čip 500 MHz VideoCore VI umožní přehrát 4Kp60 video a zobrazení na dvou monitorech, avšak pouze jeden v rozlišení 4k. Napájecí konektor je USB-C a operativní napájení je 5 v stejnosměrného napětí a maximální proud 3 A. [14]

Konektivitou je WiFi, Bluetooth a Gigabit Ethernet a 2x USB 2.0 + USB 3.0. WiFi) obsahuje protokol IEEE 802.11.b/g/n/ac, díky kterému zvládne i 5 GHz pásmo a přenosovou rychlost až 600 Mbit/s [15]. Standardně se používá 40 pinové GPIO header. Pro připojení obrazovky je k dispozici 2x microHDMI 2.0 konektor, který zvládne výše zmíněné 4Kp60 [14].

Ve výkonnosti porovnám RPi4 a starší model RPi3. Na obrázku 3.4 je porovnání obou verzí, kde je znatelně vidět, že nové Raspberry Pi 4 je přibližně o polovinu rychlejší než jeho předchůdce. Na obrázku 3.4 je porovnání i platformy a frameworku, na kterých je inference vyzkoušena. O Coral USB je napsáno v sekci 3.3.2.[16]



Obrázek 3.4 – Čas [ms] inference pro RPi3 (modrá) a RPi4 (zelená) [16]

Coral

Coral je další vývojová deska, která je od společnosti Google a vyšla v prvním čtvrtletí roku 2019. Patří k novějším zařízením s prodejní cenou okolo 3500 Kč. Coral se skládá z vývojové desky a takzvaného Coral System-on-Module. Díky tomu, že Coral je modul, je možné ho použít v prototypování a poté aplikovat do hotového produktu/systému. Dalším produktem je Coral TPU jako USB akcelerátor, který je v kapitole 3.3.2. Google udává, že inference může probíhat s modelem jako je MobileNet v2 při 400 snímcích za sekunda [17]. Co není lákavé je to, že Coral nedisponuje grafickou nadstavbou pro Linux nebo jiný operační software, a tudíž se ovládá výhradně přes příkazovou řádku nebo konzoli.

Coral je prodáván v jediné verzi. Je osazen procesorem Arm Cortex-A53 o frekvenci 1,5 GHz s grafickým čipem Vivante GC7000Lite se schopností zpracovat 1,6 GPx/s a 32 GFLOPs. Má pouze 1 GB RAM, takže při použití jiných modelů nebude dostačující. Zpracování videa má na starost VPU, která je schopna zpracovat 2 kamery ve 4Kp60 [17]. Konektivita je Wi-Fi802.11a/b/g/n/ac 2.4/5GHz, Bluetooth s podporou rozhraní UART, 2x USB 2.0 + USB 3.0, Gigabit Ethernet, 2x UART, 2x I2C, 2x SPI, 16x GPIO a 4x PWM linky. Za speciální parametry, které má, jako například kryptografický koprocesor pro asymetrické šifrování/podpisy veřejných i soukromých klíčů. 1GB LPDDR4 SDRAM s maximální frekvencí 1600 MHz, po připojení na vývojovou desku má ještě 40 GPIO (General-Purpose Input/Output) header [17].

Jak již bylo zmíněno, díky TPU, samostatně nemá v inferenci konkurenci. Na obrázku 3.5 je porovnání hodnot rychlosti vyhodnocení neuronových sítí. Jeho nevýhodu lze přiřadit k tomu, že je to specializovaný hardware, o kterém se zmiňuji v kapitole 3.3.2 a není to samostatný programovatelný hardware. [16]

Nvidia Jetson Nano

Jetson Nano je ze série vývojových desek od společnosti NVIDIA. Dále jsou k dispozici Jetson TX1, TX2 a Jetson AGX XAVIER. AGX Xavier stojí okolo 15000 Kč, TX1, které vyšlo v roce 2016, stojí okolo 8000 Kč. Jeho nástupce Jetson TX2 je za stejnou cenu až dvojnásobně výkonnějším hardwarem. Nano lze koupit v provedení vývojové desky a nebo samostatného modulu, jako je to u Google Coral, za cenu okolo 2300 Kč pro vývojovou desku a 3500 Kč pro samostatný modul. Je podobný Raspberry Pi, ale

Model architecture	Desktop CPU*	Desktop CPU * + USB Accelerator (USB 3.0) *with Edge TPU*	Embedded CPU **	Dev Board † *with Edge TPU*
MobileNet v1	47 ms	2.2 ms	179 ms	2.2 ms
MobileNet v2	45 ms	2.3 ms	15 ms	2.5 ms
Inception v1	92 ms	3.6 ms	406 ms	3.9 ms
Inception v4	792 ms	100 ms	3,463 ms	100 ms

** Desktop CPU: 64-bit Intel(R) Xeon(R) E5-1650 v4 @ 3.60GHz*
*** Embedded CPU: Quad-core Cortex-A53 @ 1.5GHz*
† Dev Board: Quad-core Cortex-A53 @ 1.5GHz + Edge TPU

Obrázek 3.5 – Porovnání Coral Dev Board proti stolnímu počítači za použití pouze CPU s TPU a bez TPU [18]

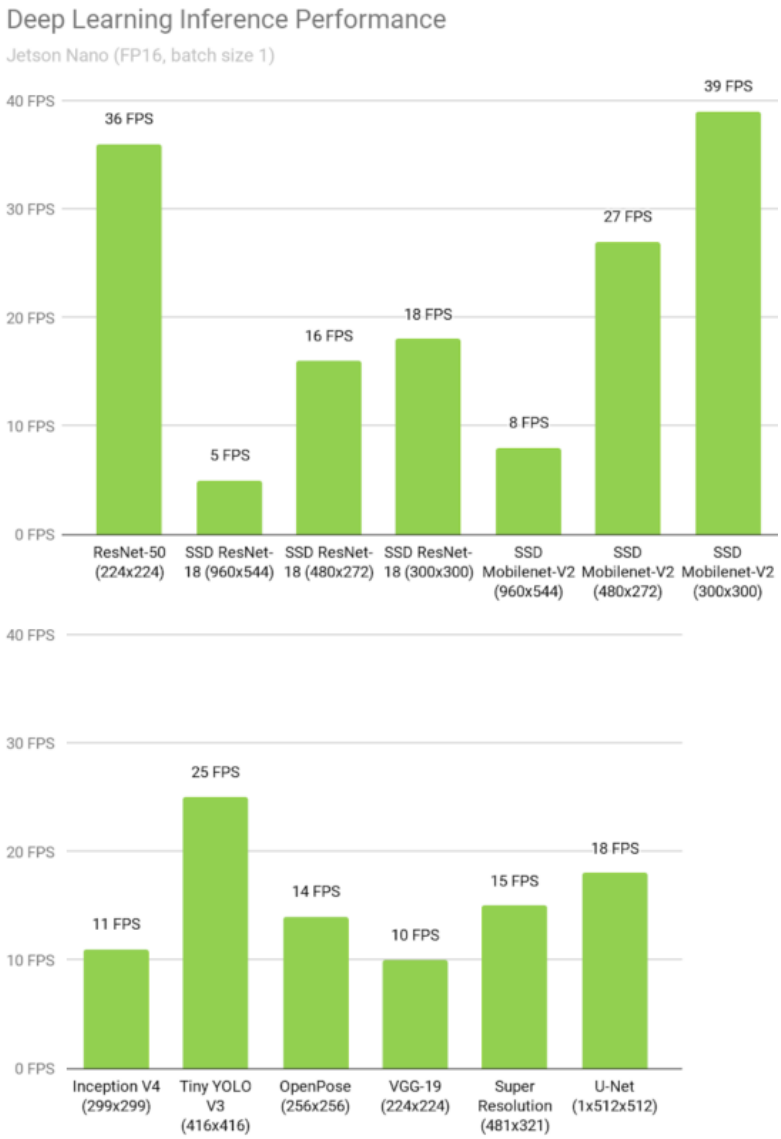
má větší výkon a disponuje kvalitním grafickým čipem [19]. Nvidia dále vlastní celý podpůrný ekosystém pro své Jetson produkty. Tento ekosystém se jmenuje Isaac SDK a Isaac SIM. Isaac je platforma pro vytváření AI (Artificial Inteligence) v robotice [20]. Při vývoji programu je ekosystém[2] velice užitečný, neboť je zaručena kompatibilita jednotlivých nástrojů pro vývoj.

Nano je prodáváno s GPU 128-core Maxwell, CPU Quad-core ARM A57 1,43 GHz s 4 GB 64-bit LPDDR4 pamětí. Jako úložiště se používá microSD karta, u samostatného modulu je přítomná 16 GB eMMC 5.1 flash pamětí.

Konektivita je Gigabit Ethernet a M.2 E pro připojení dalších úložišť jako NVMe. Disponuje rozhraním pro kameru MIPI CSI-2, schopností dekódovat videa ve 4Kp30 dvakrát, nebo 4Kp60 jednou. Obsahuje 40 GPIO pinů s podporou IIC, IIS a UART. [21]

Díky CUDA jádrům a vysoké RAM paměti je Jetson Nano velmi spolehlivým a versatilním zařízením. Na obrázku 3.6 je vidět rychlost inference neuronové sítě ve snímcích za sekundu. Přepočtem $time = \frac{1000}{FPS}$, kde FPS jsou snímky za sekundu, získáme čas inference pro porovnání s ostatními platformy [21].

[2]Ekosystémem je rozuměn soubor vývojářských nástrojů fungující v uceleném prostředí s možností použití dostupných softwarových artefaktů.



Obrázek 3.6 – Inference ve snímcích za sekundu pro Jetson Nano a ostatní platformy [21]

3.3.2 USB akcelerátory

USB akcelerátor je hardware specializovaný na inferenci neuronových sítí. Specializovaným hardwarem se nazývá ASIC (Application-Specific Integrated Circuit), v překladu Integrovaný Obvod pro Konkrétní Aplikaci. v roce 2016 na konferenci Google I/O prohlásil, že využívá TPU (Tensor Processing Unit) ve svých datacentrech již po dobu jednoho roku [22]. TPU je navržený jako maticový procesor, který v sobě má velkou matici vzájemně propojených členů pro základní aritmetické úkony jako je sčítání, odečítání, násobení a dělení a z toho důvodu je velmi rychlý při výpočtu maticových operací. TPU vykonává maticové operace s možností vysoké propustnosti dat a mnohem menší potřebou přístupu k paměti. Google vyvinul TPU pro použití

platformy pro umělou inteligenci jménem TensorFlow. [23] [24]

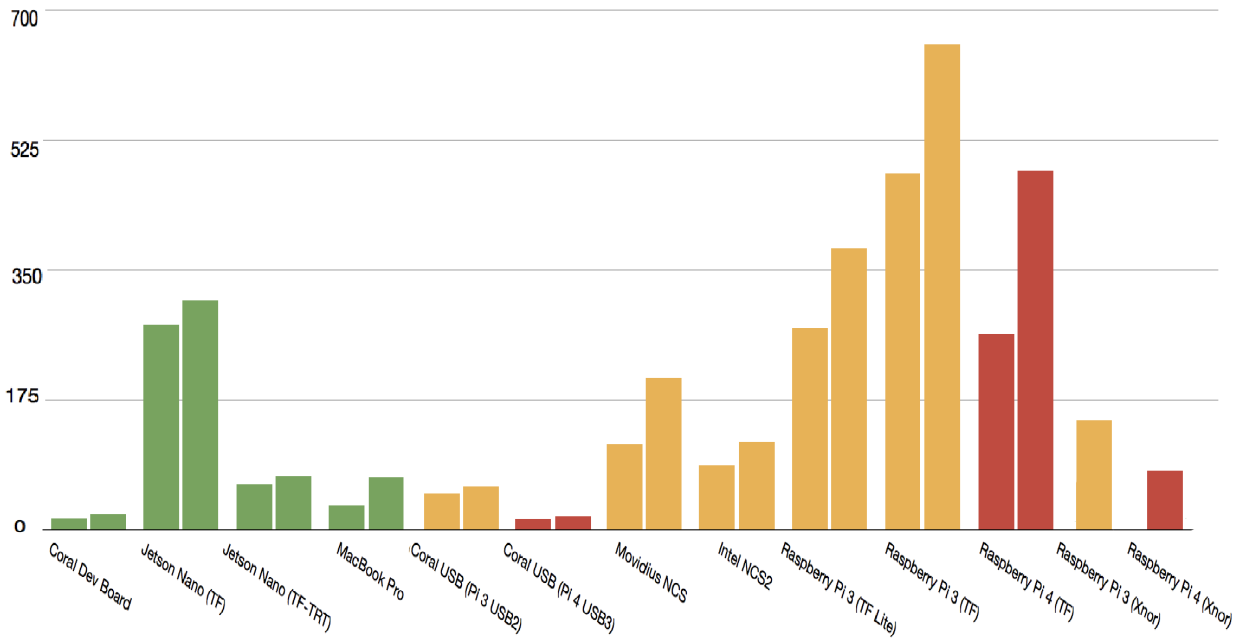
Jako první přišel na svět akcelerátor jménem Neutral Compute Stick firmy Movidius, kterou odkoupila firma Intel v roce 2016 [25].

Neural Compute Stick

Neural Compute Stick je USB akcelerátor, který existuje již delší dobu. Obsahuje procesor Intel Movidius Myriad X Vision Processing Unit (VPU). Podporované frameworky jsou TensorFlow, Caffe, Apache MXNet, Open Neural Netowrk Exchange (ONNX), PyTorch a PaddlePaddle skrze ONNX converzi s rozhraním USB 3.0. Podporované operační systémy jsou Windows 10, Raspbian, Ubuntu 16 a CentOS 7.4. Cena za kus je okolo 1600 Kč. [26]

Coral USB Akcelerátor

Firma Google ohlásila Coral USB akcelerátor v létě v roce 2018 a vyšel v březnu v roce 2019 [27]. Nejnižší sloupce ve zmíněném grafu 3.7, jsou časy inferencí pro Coral Dev Board a Coral USB s použitím RPi4. Jelikož časy jsou téměř shodné a RPi4 samostatně nedosahuje vynikajících výsledků jako TPU. z toho můžeme usuzovat, že v Coral USB akcelerátoru se používá stejné TPU jako se používá ve vývojové desce. Cena je okolo 1800 Kč [28].



Obrázek 3.7 – Čas [ms] inference modelu MobileNet v1 proti v2 (vlevo v1; vpravo v2) [16]

3.3.3 Vyhodnocení vývojových desek

Konkurence mezi třemi zmíněnými vývojovými deskami je obrovská. Na sloupcovém grafu 3.7 je vidět, že Coral Dev Board a Coral USB je nejrychlejší v inferenci. To je dáno použitím TPU, které je vyrobeno za účelem rychlé inference. Jetson lze použít s frameworky bez potřeby dodatečného hardwaru, kombinace Raspberry Pi s akcelerátorem je možné spustit ve všech frameworcích a Google Coral je odkázaný na použití frameworků a modelů z jejich dílny [28].

Raspberry Pi disponuje velmi nízkou cenou, Jetson Nano svojí versatilitou a Coral svojí rychlostí pro vyhodnocování neuronových sítí díky TPU procesoru.

Tabulka 3.2 – Tabulka hardwarových specifikací

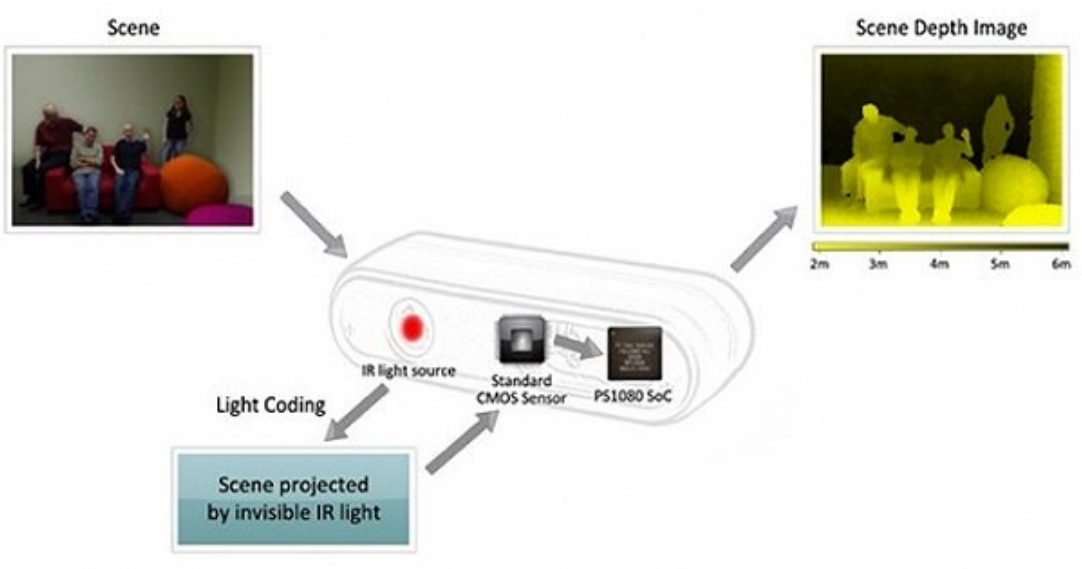
	Jetson Nano	Coral Dev Board	RPi4(4 GB) + Coral USB
Procesor	Quad-core ARM A57 1,43GHz	Quad-core ARM A53 1,5 GHz	Quad-core ARM Cortex-A72 1,5 GHz
RAM	4GB 64-bit LPDDR4	1 GB LPDDR4	4GB LPDDR4
Grafický čip	GPU 128-core Maxwell	Vivante GC7000Lite	500 MHz VideoCore VI
Gigabit Ethernet	ANO	ANO	ANO
Wi-Fi	X	ANO	ANO
Bluetooth	X	ANO	ANO
Čas inference (ref. modelu MobileNetV2)	25,64 ms	2,3 ms	2,3 ms
Cena [Kč]	2300	3500	1500 + 1800 (3300) height

3.3.4 Kamery se snímáním vzdáleností

Kamery se snímáním vzdálenosti fungují na principu vysílání monochromatického IR rastru do snímaného směru, kde snímač tento rastr zachytí a vypočítá vzdálenost podle času, kterou paprsek cestoval. [29]

Intel RealSense

Intel RealSense je kamera s širokou podporou programovacích jazyků a frameworků. Je kompatibilní s enginy jako je Unreal Engine a Unity pro rekreaci 3D objektů ve virtuálním prostředí (možné i pro VR (Virtual Reality)). Má podporu programovacích jazyků C++, MATLAB i jazyka Python a existuje i balíček pro ovládání přes ROS (Robot Operating System). Jako technické specifikum má RGB senzor až 1920x1080 pixelů se schopností snímat 30 snímků za sekundu v tomto rozlišení. Minimální vzdálenost, kterou dokáže snímat je 16 centimetrů v rozlišení 1280x720 pixelů s 90 snímky za sekundu. Maximální vzdálenost podle typu je přibližně 10 metrů [30].



Obrázek 3.8 – Jednoduché schéma snímání vzdálenosti kamerou [29]

Microsoft Kinect

Microsoft má své vlastní kamery s měřením hloubky používané u herních konzolí Xbox. Existuje oficiální API (Application Programming Interface) pro detekování pózy člověka, ale je omezené pouze na jazyk C++ [31]. Ke kameře se může stáhnout balíček OpenKinect, který vyvíjí skupina lidí v zájmu použití Kinectu na ostatních platformách než je Windows. Existují k němu i wrappery na Python, Javu i C++ [32]. Jeho technická specifika jsou následující. Maximální rozlišení, které je schopné zachytit je 1280x960 pixelů s 15 snímky za sekundu a jeho snímač vzdálenosti je schopný zachytit v rozsahu 50 cm až 5 metrů s rozlišením 640x480 pixelů. Dodatečně obsahuje i mikrofon [33].

3.4 Software

3.4.1 Programovací jazyky a framework

Python

Python je programovací jazyk, který je velmi populární zvlášť v posledních letech. Díky tomu, že je to vysokoúrovňový jazyk a připomíná strukturovanou angličtinu, se v něm lze velmi rychle naučit. Je to open source projekt, který nabízí zdarma své instalační balíčky pro jeho používání. Existují tři verze Pythonu. První verze se již nepoužívá a byla vydána v roce 1991. Pro druhou verzi bylo letošní rok vyhlášeno konec podpory a vývoje, tudíž poslední, třetí verze je aktivní a používá se. Python má velmi širokou

základnu uživatelů a spousta firem vytváří i své open source balíčky, které může použít každý.

Robot Operating System

Robot Operating System, je open source framework pro vytváření programů pro roboty. Zahrnuje velkou řadu nástrojů a knihoven, které mají usnadnit vytváření komplexních a robustních programů pro roboty napříč mnoha platformami. ROS byl vytvořen pro kolaborativní vývoj softwaru. Například laboratoř má znalce na mapování interiéru a mohou přispět do světového ekosystému svým produktem. Jiná skupina využívá mapy pro navigaci a další vymyslela algoritmus pro počítačové vidění pro rozpoznávání malých předmětů ve shlucích [34]. Tímto může každý využít tyto příspěvky, a tak vytvořit komplexní program pro robota. ROS podporuje programovací jazyky C a Python.

3.4.2 Simulační software

Simulace robota je při vývoji užitečná hlavně z důvodu, kdy si člověk ověří funkčnost napsaných algoritmů v softwarovém prostředí, a tím nepotřebuje reálné díly, které by se mohly poškodit. Simulace se běžně využívá již spoustu let, ale dnes je velmi moderní a žádané mít digitální dvojče díky iniciativě průmyslu 4.0. Tyto simulátory jsou pro verifikaci a validaci návaznosti kinematiky, které zahrnují základní fyzikální zákony.

Gazebo

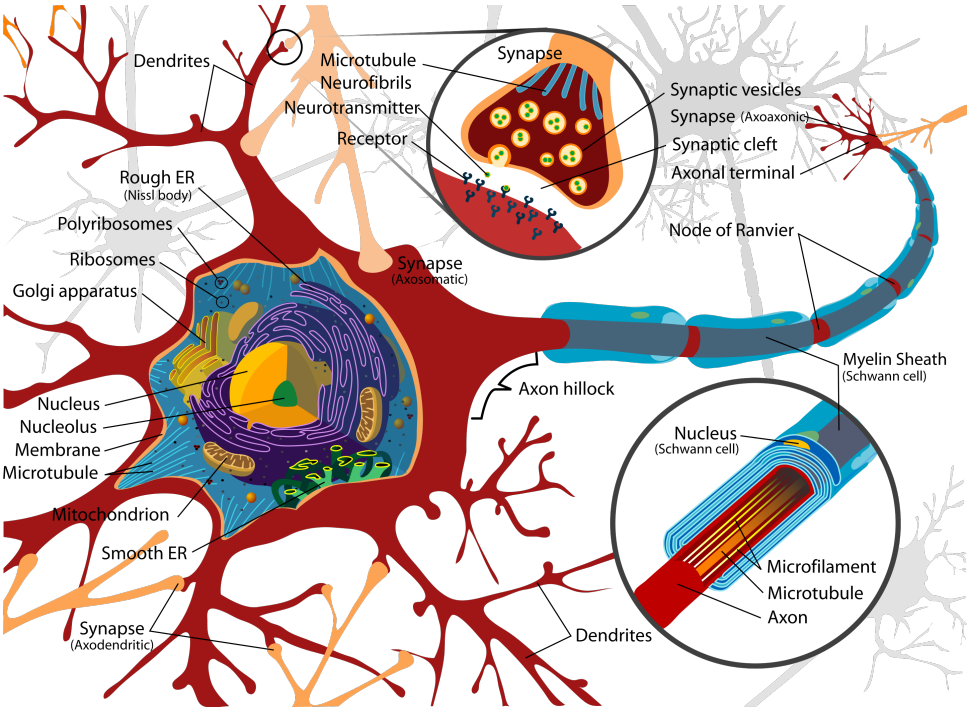
Gazebo je 3D simulační software, který je často spojován přímo s ROS. Počítá s fyzikou, mohou se doinstalovat dodatečné balíčky, modelovat roboty s jejich kinematikou za pomocí SDF (Simulation Description Format). Lze v něm i generovat data ze senzorů s šumem. Mezi tyto senzory patří například 2D a 3D kamery, kontakntní sensory, Kinect, laserové měřicí zařízení a další. Používá se s jazykem C++. [35]

Webots

Webots je alternativní řešení ke Gazebu a je též open source. ve své podstatě je velmi podobný zmíněnému Gazebu. Obsahuje vše, co obsahuje Gazebo, ale podporuje více programovacích jazyků, krom C a C++ se v něm může programovat i v Javě, Pythonu a Matlabu.

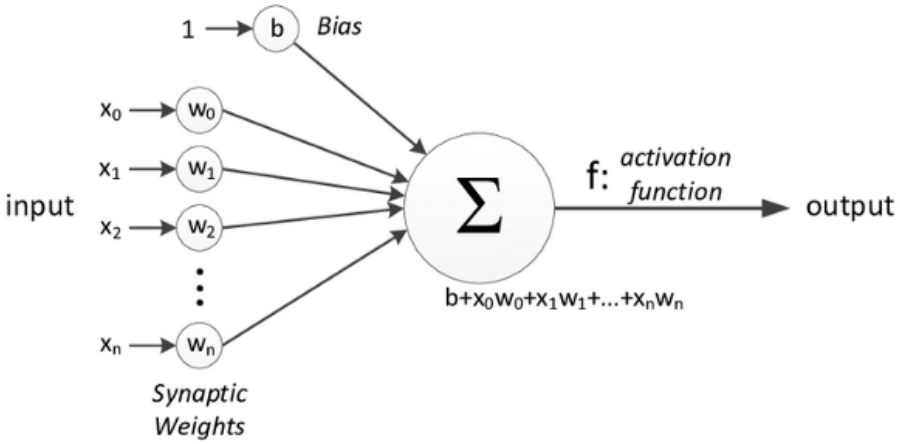
3.5 Umělé neuronové sítě

Neuron je anatomická jednotka nervové soustavy, která je schopná přijmout druh signálu, odpovědět na něj jiným signálem a vytvářet funKční kontakty (synapse) s jinými neurony. Na obrázku 3.9 je vidět anatomický popis neuronu, nejdůležitějším prvky pro vysvětlení analogie k umělé neuronové síti jsou dendrity a axon. Dendrity jsou částmi neuronu, které přijímají informace. Informace ze všech dendritů jsou zpracovány neuronem a další informace je poslána na výstup do axonu. Axon je napojen na další neuron, vytváří synapsi a je prvkem neuronové sítě [36] [37].



Obrázek 3.9 – Neuron [36]

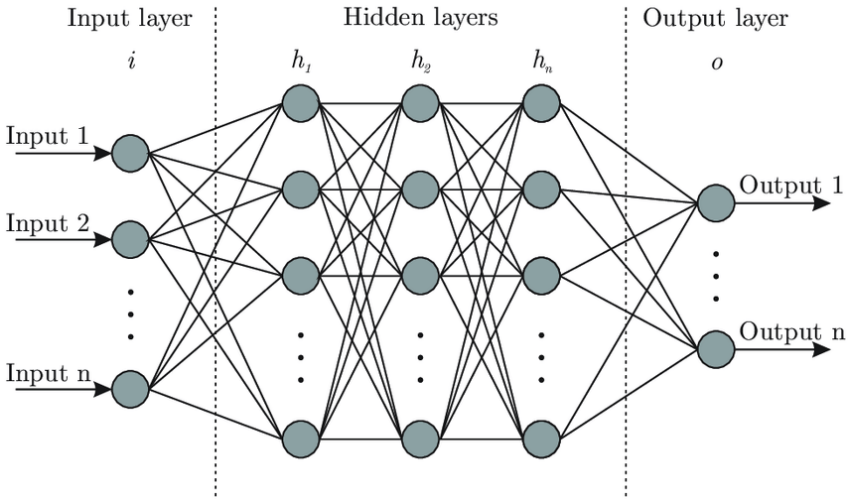
Umělá neuronová síť byla inspirována dle výše popsaného principu. Nejmenší část umělé neuronové sítě je perceptron (obr. 3.10). Vstup do perceptronu je vyhodnocován sumou vstupním vektorem $\vec{X}$ vynásobený vektorem synaptických vah $\vec{W}$ s přičtenou konstantou b. Aktivační funkce je vyhodnocovací funkce, která přemění vstupní hodnotu na výstupní. Je-li nelinerání, poté je neuronová síť schopná řešit i nelineární problémy [37].



Obrázek 3.10 – Perceptron [37]

Neuronová síť se poté skládá z mnoha perceptronů uspořádaných do vrstev. Do první vrstvy vstupují data, která mají být natrénována. Poslední vrstva je akce, která má být vykonána - například při rozpoznání obrázků, které jsou klasifikované do tříd, je výstupem její třída. Mezi těmito vrstvami jsou skryté vrstvy, ve kterých jsou neznámé hodnoty, které se upravují podle cenové funkce. Příklad neuronové sítě lze vidět na obrázku 3.11.

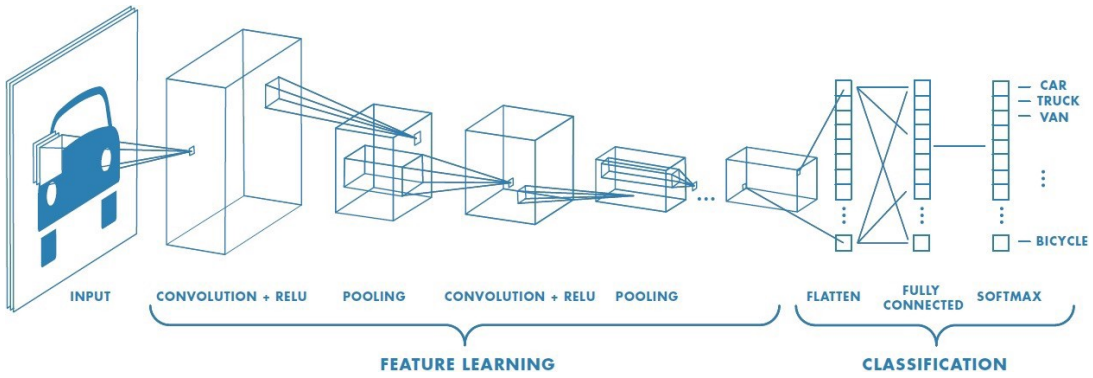
Cost function je metrika, podle které se určuje přesnost a správnost výsledku vyhodnocená neuronovou sítí a na jejím základě je neuronová síť schopná se učit - adaptovat váhy. Ve vrstvě se její váhy upraví dle hodnoty učícího algoritmu.



Obrázek 3.11 – Neuronová síť [38]

3.5.1 Konvoluční neuronová síť

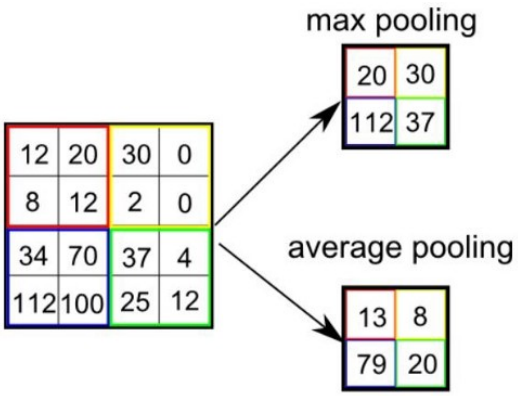
Konvoluční neuronová síť (CNN) je podmnožinou neuronových sítí, které se používají především při učení rozpoznání obrazu. Jejich hlavní charakteristika je použití konvolučních a poolingových vrstev, jak je uvedeno na obrázku 3.12. Každý obrázek je zastoupen maticí pixelů s hodnotou jasu, v případě barevných obrázků obrázku 3 kanály - RGB.



Obrázek 3.12 – Schéma konvoluční neuronové sítě [39]

V konvoluční vrstvě probíhá konvoluce dle rovnice (3.15), kde f je snímek, na kterém je prováděná konvoluce a h je konvoluční filtr, kterému se také říká kernel. Většinou je reprezentován jako matice 3x3 s pevně danými hodnotami. Podle hodnot uvnitř kernelu způsobí například vyhlazení - rovnice (3.16), doostření - rovnice (3.17), detekci hran - rovnice (3.18), a jiné [40]. Konvoluční vrstva je schopná vyextrahovat znaky důležité pro daný obrázek, díky kterým se celá síť dokáže naučit rozpoznat, co na obrázku je.

$$f(x,y) \circledast g(x,y) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} f(x-i, y-j)\, h(i,j) \tag{3.15}$$



Obrázek 3.13 – Poolingová vrstva [39]

Vstupní snímek prochází různými konvolučními filtry a tím znásobuje data. Poolingová vrstva slouží pro redukci parametrů, které vzniknou konvolucí. Je to operace, ve které se vezme velikost poolingového filtru NxN. Tato matice je následně rozložena rovnoměrně po snímku, jako je vyobrazené na obrázku 3.13. Do další vrstvy postupuje pouze jedna hodnota z vybraného regionu poolingového filtru. Tato hodnota je zvolena jako hodnota v příslušném filtru buď maximální (Max Pooling), průměrná (Average Pooling) nebo součtová (Sum Pooling). [39]

$$\begin{bmatrix} 0,0625 & 0,125 & 0,0625 \\ 0,125 & 0,25 & 0,125 \\ 0,0625 & 0,125 & 0,0625 \end{bmatrix} \tag{3.16}$$

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix} \tag{3.17}$$

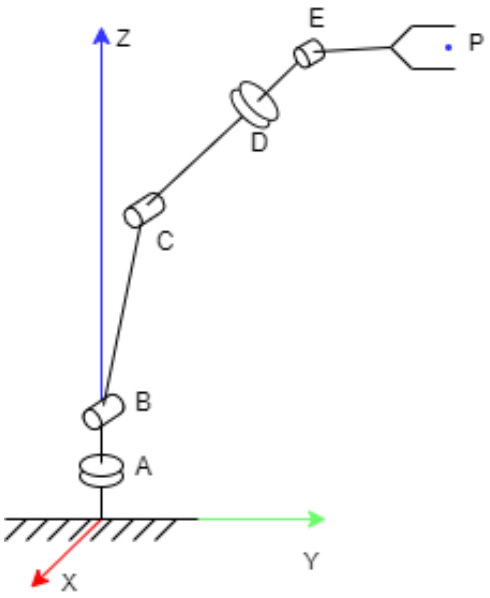
$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{3.18}$$

Kapitola 4

Praktická část

4.1 Struktura manipulačního zařízení

Jelikož mám v plánu použít balíček pro rozeznání postury člověka, zvolil jsem si strukturu připodobněnou lidské paži. Konstrukce je sestavena z kinematických dvojic, ramenní kloub nahradím dvěma rotačními vazbami, loketní kloub s jedním stupněm volnosti, kde v předloktí bude rotační vazba souběžná s osou loketní kosti pro dosažení rotace zápěstí. Na loketní kloub bude navazovat zápěstí s jedním stupněm volnosti a jeho koncový efektor. Znázornění je na obrázku 4.1. Rotační vazby označenými body A a B nazývám ramenní kloub, bod C nazývám loketní kloub, vazbě D předloktí a vazbě E zápěstí. Částem mezi body B a C nazývám paže, mezi body C a E předloktí a body E a P ruka.

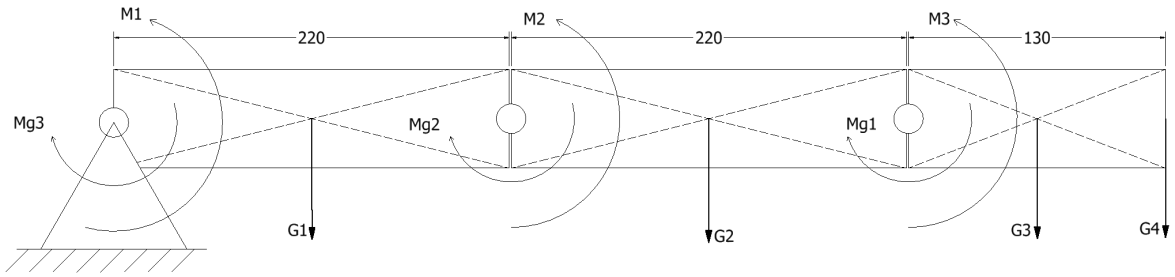


Obrázek 4.1 – Struktura manipulačního zařízení

4.2 Konstrukční návrh manipulačního zařízení

V konstrukčním návrhu se zaměřuji na proveditelnost s ohledem na případnou realizaci technologií 3D tiskem. Maximální tiskový rozměr, který mám k dispozici je 220 mm, proto budu robota vytvářet s ohledem k těmto rozměrům. Budu vycházet z počátečního odhadu a dimenzování servopohonů tak, aby manipulační zařízení uzvedlo zatížení alespoň půl kilogramu.

Jako počáteční odhad jsem zvolil rozměry 220 mm pro paži, 220 mm pro předloktí a 130 mm pro ruku. Uvažoval jsem homogenní kvádry z materiálu PLA, jehož hustota je 1250 kg/m^3. Rozměr podstavy kvádru jsem uvažoval 50 mm na 50 mm a jednoduchým výpočtem jsem zjistil jednotlivé momentové síly, které jsou v tabulce 4.1. Z vypočtených hodnot v této tabulce jsem uvažoval potřebné momenty pro dimenzování servopohonů v kloubech. Momenty byly počítány v natažené pozici, jako je to na obrázku 4.2. Podmínkou dimenzování je vyjádřena rovnicí 4.1.



Obrázek 4.2 – počáteční výpočty
M1, M2, M3, M4 - točivé momenty motorů
Mg1, Mg2, Mg3, Mg4 - momenty síly vyvinuté tělesy
1 - paže, 2 - předloktí, 3 - ruka, 4 - zátěž

Tabulka 4.1 – Tabulka výpočtů

Těleso	Délka [m]	Hmotnost [kg]	Momentová síla [Nm]	Kumulativní momentová síla [Nm]
Paže	0,22	0,69	0,74	2,38
Předloktí	0,22	0,69	0,74	1,64
Ruka	0,13	0,41	0,26	0,90
Zátěž	0,13	0,50	0,64	0,64

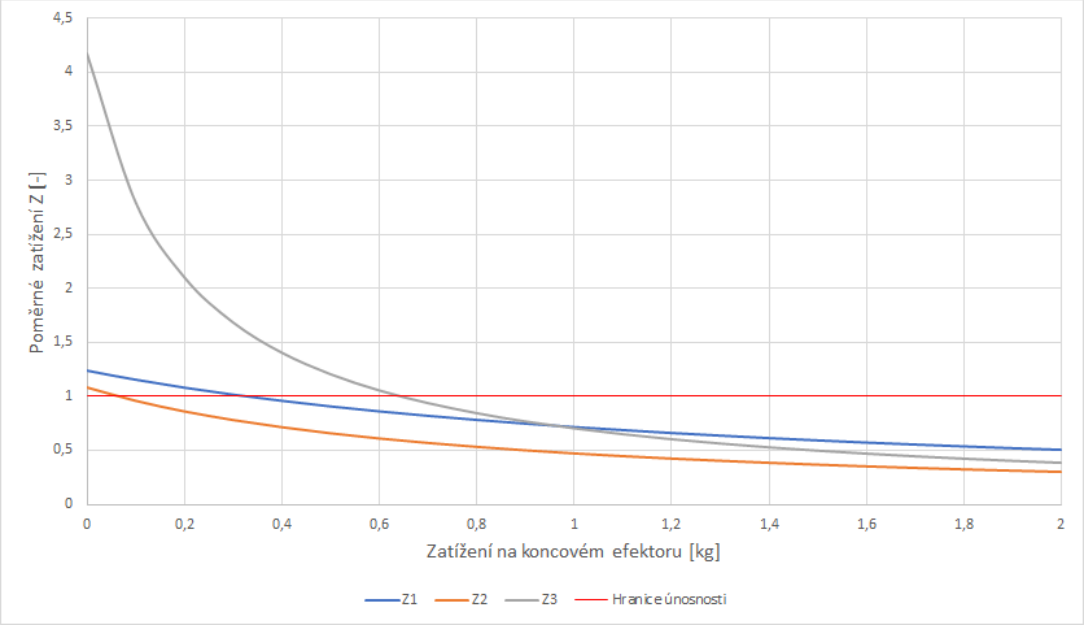
Pro dimenzování jsem využil rovnici 4.1, kde M_z je moment od zátěže na koncovém efektoru.

$$\begin{aligned} M_1 > M_2 > M_3 > M_z \wedge M_1 > Mg_1 + Mg_2 + Mg_3 + M_z \wedge \\ \wedge M_2 > Mg_2 + Mg_3 + M_z \wedge M_3 > Mg_3 + M_z \end{aligned} \tag{4.1}$$

Točivé momenty servopohonů se udávají v jednotkách $kgcm$, kde $1\ kgcm = 0,098\ Nm$ a nadále je budu vždy přepočítávat na jednotky newton metrů.

Pro počáteční odhad točivého momentu servopohonů jsem zvolil pro rotaci v ramenním kloubu dva servopohony MG996R, jejichž točivý moment je 1,078 Nm. V konstrukci tím pádem musím uvažovat zátěž na druhé straně kloubu k dovážení momentů, abych splnil podmínku dimenzování. Pro jednoduchost budu v loketním kloubu budu uvažovat dva stejné servopohony a do zápěstí vložím pouze jeden zmíněný servopohon. Získávám tedy točivé momenty $M_1 = 2,156\ Nm$, $M_2 = 2,156\ Nm$ a $M_3 = 1,078\ Nm$. Zatížení na koncovém efektoru je proměnlivé a bude mít proporcionální vliv na každý kloub. Pro ověření dimenzování jsem graficky vykreslil poměrové

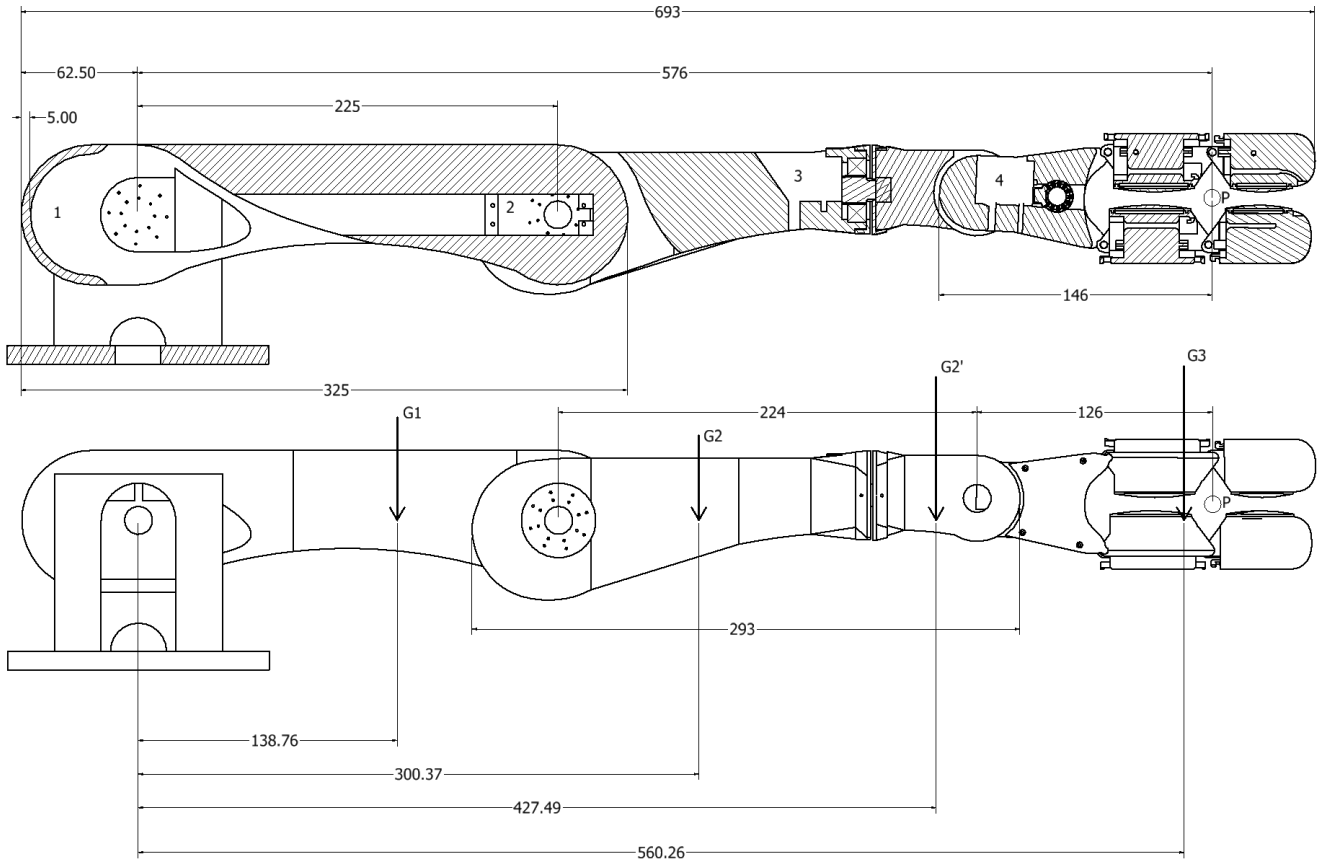
zatížení $Z_n = \frac{M_n}{Mg_n}$, kde M_n je moment motoru a M_{gn} je moment od tíhového zrychlení a jejich indexy těles $n \in \{1,2,3\}$, do kterého je započítán vliv koncového efektoru. Výsledek je vidět na obrázku 4.3. Jelikož je kinematika sériová, tak nejvyšší zatížení, které bude moci uzvednout je to místo protnutí, která momentová křivka protne hranici únosnosti jako první. Z grafu je patrné, že úzkým místem je zatížení v lokti a celý systém bude schopné unést svojí hmotnost. Tímto základním propočtem jsem si vědom toho, že konstrukci musím udělat lehčí, použít vyvažování a nebo nadimenzovat servopohony s vyšším točivým momentem.



Obrázek 4.3 – Poměr momentu vyvinutý servopohony ku zatížení vlastní vahou v přibližném výpočtu

4.2.1 Prvotní konstrukční návrh

Před aktem konstruování je obtížné odhadnout, jak daleko od kloubu bude těžiště či jakou konečnou hmotnost těleso bude mít, neboť se již nepočítá s homogenním kvádrem jako při odhadu. V prvotní konstrukci jsem na osovou vzdálenost dbal, včetně prostoru pro vyvážení. Cílem bylo udělat konstrukci co možná nejlehčí, ukrýt servopohony pokud možno do konstrukce, aby design byl z hlediska vodičů čistý. Na obrázku 4.4 je vidět výsledek se zakótovanými těžišti v místech se šipkami a v tabulce 4.2 jsou vidět další údaje. Označení G_1, G_2, G_2', G_3 jsou tíhy jednotlivých těles. Hmotnosti a těžiště byly spočítány programem Autodesk Inventor, ve kterém jsem celého robota konstruoval. V návrhu jsem se snažil respektovat vše z předchozí části této kapitoly.



Obrázek 4.4 – Okótovaný první návrh konstrukce s těžišti
G1 až G3 - těžiště
1 - komora pro vyvážení, 2 - komora pro servopohon v loketním kloub
3 - komora pro servopohon pro předloketní rotaci, 4 - komora pro servopohon pro koncový efektor
P - bod pro chytání předmětů

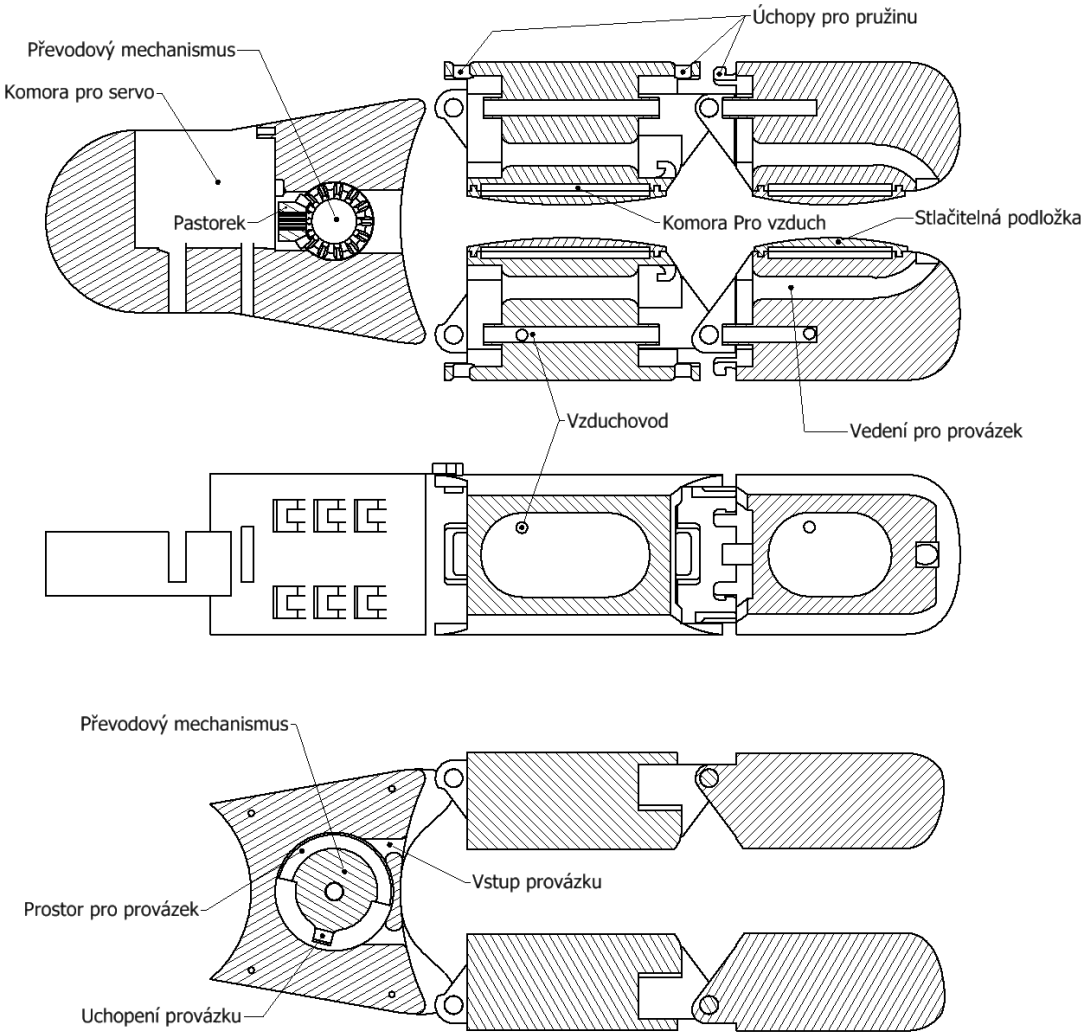
Osová vzdálenost ramenního a loketního kloubu je přibližně stejná, ale těleso je těžší, jelikož je celkově větší a při konstrukci jsem opomněl, že je přídavný materiál kolem samotné osy. V loketním kloubu jsou ukryty dva servopohony MG996R s točivým momentem 11 kg/cm při napětí 6 V [41]. Z původního záměru ukrýt servopohony do konstrukce, ovlivnilo její šířku, která se navýšila na 87 mm. Tím objem nabyl a hmotnost je vyšší než původní odhad. V ramenním kloubu moment 2,16 Nm nevyhovoval, na obrázku 4.4 je komora označená (1). Do komory se připevní závaží o určité hmotnosti pro vyvážení, proto jsem moment v této části nijak neměnil. Osová vzdálenost mezi loketním kloubem a zápěstím robota je mnohem větší. Při konstruování jsem si nedával pozor a opomněl jsem rotaci v předloktí, kterou jsem následně dodělával. Mezi zápěstím a pracovním bodem P, pro který je brán bod uchopení předmětů splňuje velikost, od osy zápěstí po konec koncového efektoru nikoliv.

Tabulka 4.2 – Tabulka výpočtů

Těleso	Těžiště [m] hmotnosti	Hmotnost [kg]	Momentová síla [Nm]	Kumulativní momentová síla [Nm]
Paže	0,138	1,152	0,97	2,39
Předloktí	0,66	0,69	0,80	1,42
Ruka	0,56	0,26	0,62	0,62

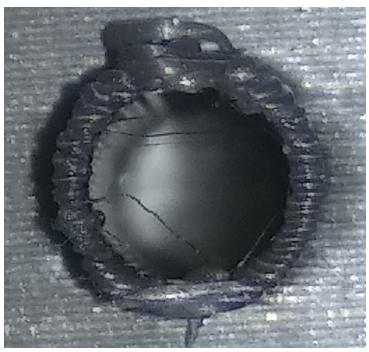
První návrh koncového efektoru

Chtěl jsem zůstat u připodobnění lidské kinematiky, rozhodl jsem se připodobnit i koncový efektor k lidské ruce. Je známo z anatomie, že sval stáhne šlachu v prstu, tím se zkrátí jeho délka a v kloubech se prst ohne. Do původní pozice by se vracel za pomocí pružin. Jako další požadavek jsem chtěl, aby ruka byla schopná mít zpětnou vazbu z dotyku. Toho lze docílit například stlačitelností vzduchu. Na obrázku 4.5 je uveden vzduchovod a vzduchová komora, jednotlivé vzduchovody článků prstů se propojí trubičkou, které by vedly následně do MEMS (Micro ElektroMechanical System) tlakového senzoru [42].



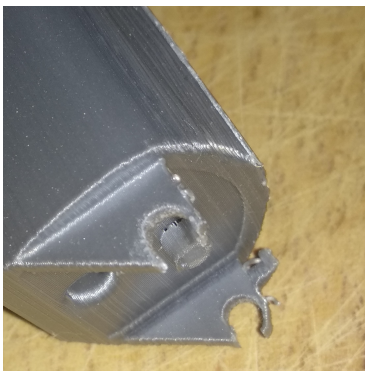
Obrázek 4.5 – Návrh koncového efektoru

Konstrukci prstů jsem si zkusil vytisknout na 3D tiskárně, ale našel jsem několik nedostatků, které vznikly samotnou povahou tisknutí na FDM tiskárně a několik závad z konstrukčního hlediska.



Obrázek 4.6 – Nerovnoměrnost tisknutí

Obrázek 4.7 – Nespojení materiálu



Obrázek 4.8 – Ulomené pacičky

Na obrázku 4.6 je vidět nerovnoměrnost kruhovitého povrchu zapříčiněná kladením jednotlivých vrstev na sebe. Tento efekt je očekávaný vzhledem k použití FDM 3D tiskové technologie. V některých místech se položila vrstva o kousek blíž k ose a jsou viditelnější propasti. Při napasování trubičky s přechodovou rozměrovou velikostí, pak musí být zajištěné těsnění pro vzduchotěsnost. Dalším řešením je změna orientace při tisknutí či zabroušení a následné vyhlazení povrchu. Pro vyhlazení povrchu je možné použít chemického rozpouštění, ale PLA je hůře rozpustitelný. Podle článku [43][1] se PLA plně rozpustí v chloroformu až za 8 dní, v tetrahydrofuranu za 12 dní a v acetonu při vyšších teplotách za 3 dny.

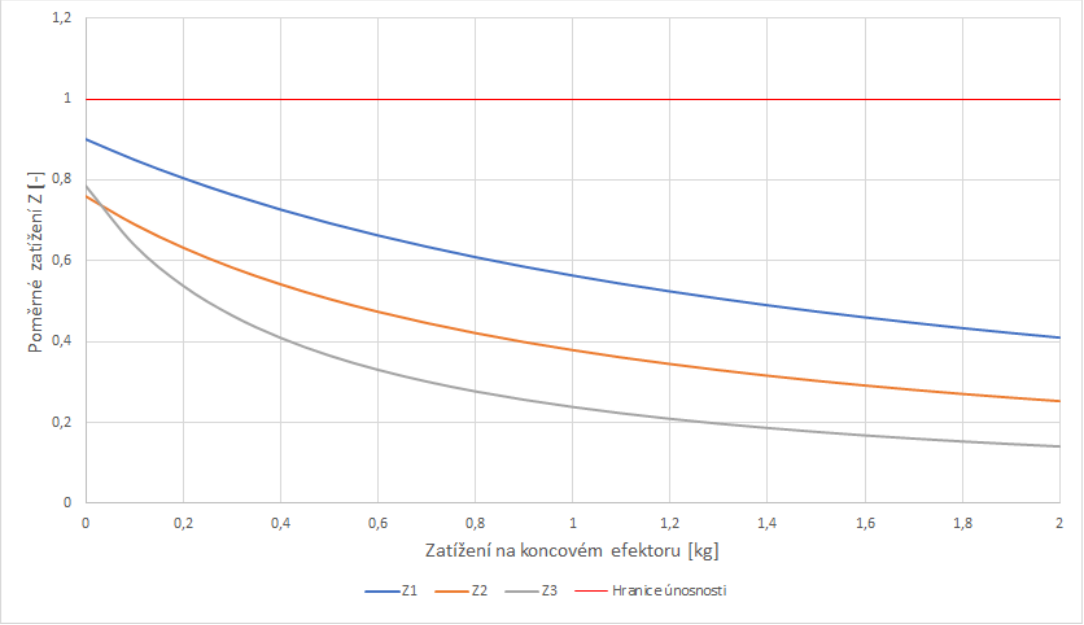
Na obrázku 4.7 je viditelné nespojení materiálu, jelikož je tloušťka stěny stejná jak jedna vrstva. Toto se může ve sliceru[2] stát a je třeba si na to dát pozor.

Na poslední obrázku 4.8, jsou viditelné ulomené pacičky pro držení konstrukce. Pacičky se ulomily po aplikování poměrně malé síly. Pevnější konstrukce v místě zlomu tento problém může vyřešit.

Celá konstrukce koncového efektoru váží 0,258 kg a bylo by nutné odlehčení celé sestavy. Z výše uvedených důvodů jsem se rozhodl celý návrh robota předělat, což je popsáno v další kapitole 4.2.2

Z grafu 4.9 pro momenty $M_1 = 22\ kgcm, M_2 = 11\ kgcm, M_3 = 5\ kgcm$ je vidět, že celý návrh konstrukce selhal. Je možnost využít silnějších servopohonů a předělat konstrukci pro silnější servopohony nebo ponechat servopohony a upravit konstrukci. Jelikož mám i maximální dostupný tiskový prostor 220 mm, vybral jsem si možnost upravit konstrukci a zanechat servopohony.

[1] Uvedená studie byla provedena ke zjištění rozpustnosti konkrétního PLA plastu zvaným Ingeo™
[2] Program pro generování G-kódu z 3D modelu pro tisknutí na 3D tiskárně



Obrázek 4.9 – Poměr momentu vyvinutý servopohony ku zatížení vlastní vahou v prvotní konstrukci

4.2.2 Vylepšení návrhu manipulačního zařízení

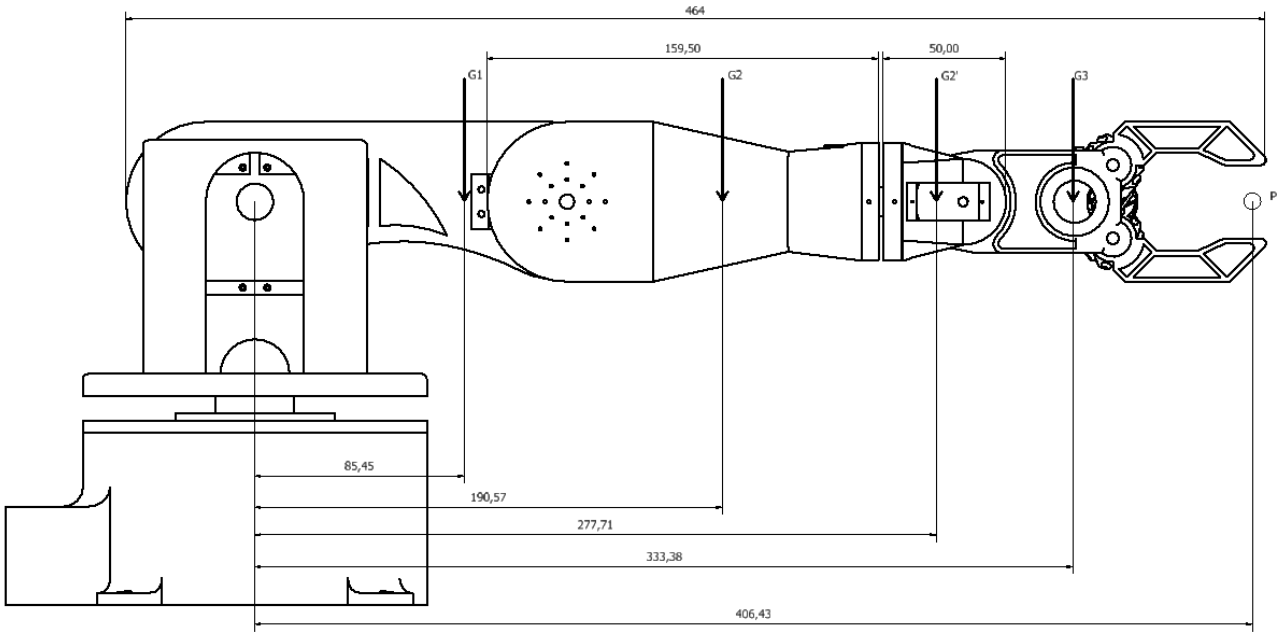
V novém návrhu jsem zkrátil osové vzdálenosti pro jednotlivé části konstrukce a upravil koncový efektor. Osovou vzdálenost v paži jsem zmenšil téměř o polovinu a v předloktí o 60 centimetrů. Vzdálenost mezi osou rotace ruky a bodu P je menší pouze o kousek. Celkovou délku celé konstrukce jsem zmenšil o polovinu a koncový efektor je tak značně lehčí. Celkové zhodnocení je vidět v tabulce 4.3.

Tabulka 4.3 – Tabulka výpočtů nový návrh oproti předchozímu

Těleso	Těžiště [m] hmotnosti	Hmotnost [kg]	Momentová síla [Nm]	Kumulativní momentová síla [Nm]
Nový návrh				
Paže	0,085	0,37	0,31	0,91
Předloktí	0,088	0,27	0,80	0,40
Ruka	0,095	0,06	0,06	0,06
Předchozí návrh				
Paže	0,138	1,152	0,97	2,39
Předloktí	0,66	0,69	0,80	1,42
Ruka	0,56	0,26	0,62	0,62

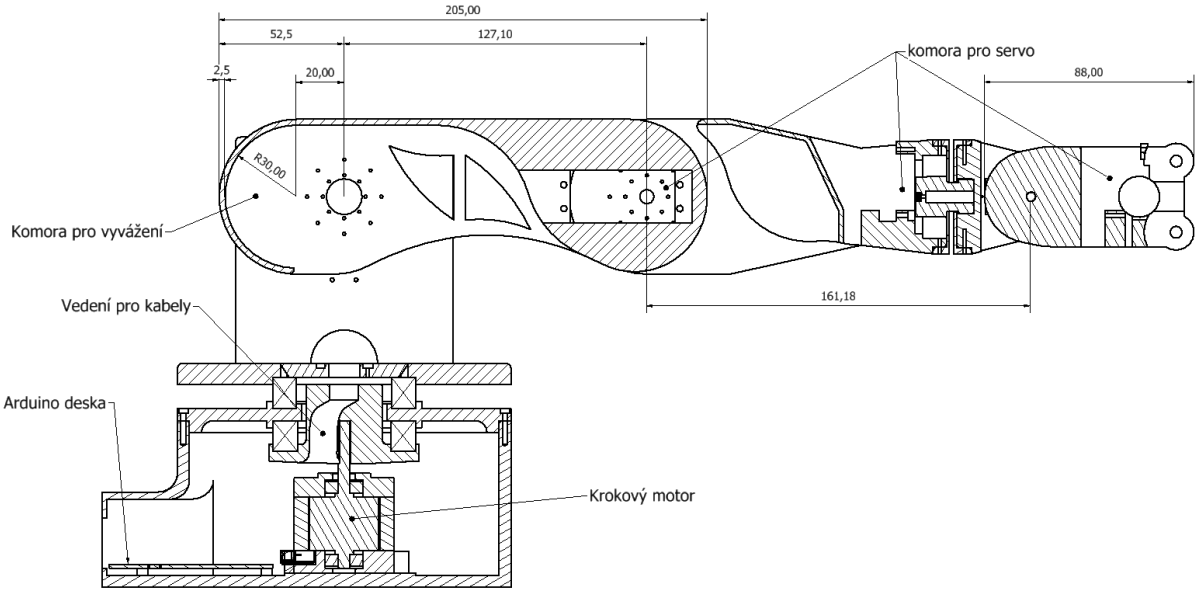
V základně robota je umístěný krokový motor NEMA 17. Přenášení točivého momentu je přes hřídel, která je z části vydutá, aby vedla kabely ze všech servopohonů k arduino základní desce, která je bude ovládat. Tímto by se mělo dosáhnout čistějšího provedení,

aby bylo uvnitř konstrukce robota schováno co nejvíce kabelů. Předpokládaný rozsah otáčení okolo osy základny je přibližně 200° na každou stranu, kde softwarově bude ošetřen úhel natočení, aby se robot netočil stále dokola a nedošlo k porušení kabelů.



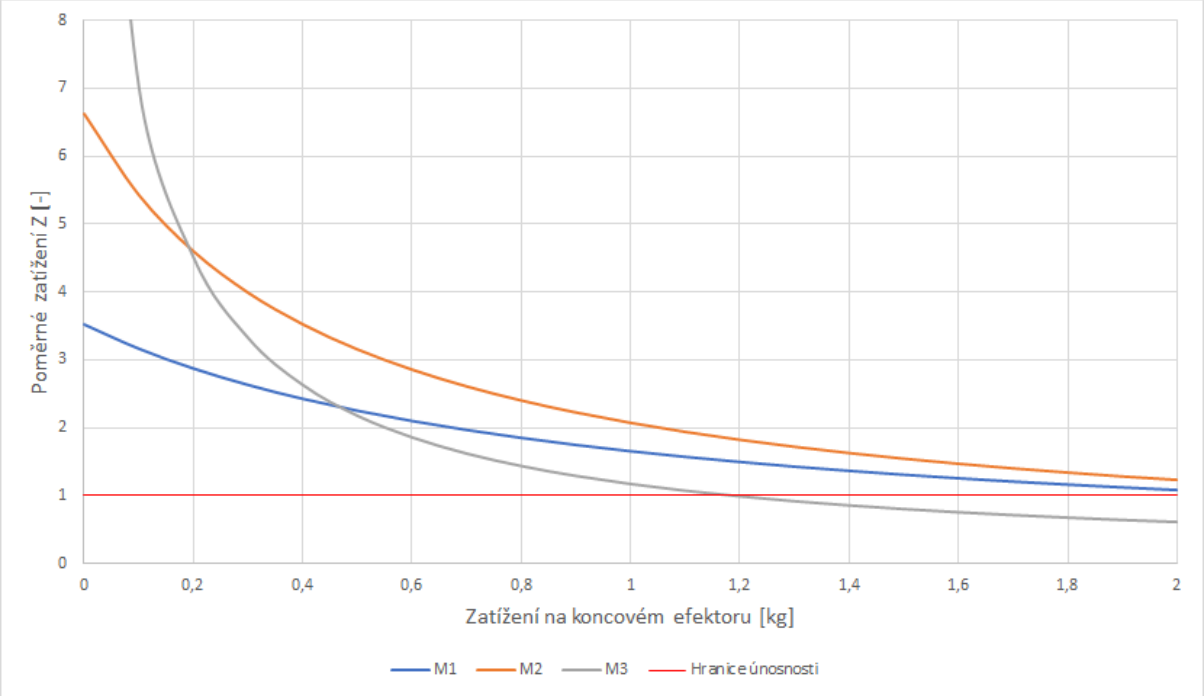
Obrázek 4.10 – Okótovaný první návrh konstrukce s těžišti
G1 až G3 - těžiště
P - bod pro chytání předmětů

Komora pro vyvážení je válcovitá o poloměru 30 mm a výškou podstavy 79 mm. V případě vylití olova do tvaru půlválce, by vyvážení mělo efekt 0,083 Nm. Návrh je v ostatních ohledech převzatý z prvotního návrhu. Konstrukce je odlehčená v místech, kde jsem dle inženýrského odhadu uznal, že se tím značně nesníží tuhost. Pevnostní analýza není předmětem této práce, a z toho důvodu ji vynechám. Robot disponuje v ramenním kloubu momentem 2,239 Nm včetně vyvažovacího závaží, kde 2,156 Nm je moment od dvou servopohonů MG996R. V loketním kloubu má též dva servopohony MG996R, zápěstí disponuje dvěma servopohony SAVOX SH-0256, s celkovým momentem 0,92 Nm. Všechny uvedené momenty jsou za předpokladu napájecího napětí 6 V, které je maximální možné.



Obrázek 4.11 – Okótovaný druhý návrh konstrukce

Na grafu 4.12 je vidět poměrové zatížení s ohledem na zatížení na koncovém efektoru. Zápěstí je opět úzké místo, které uzvedne necelých 1,2 kg. Tento návrh již považuji za úspěšný.



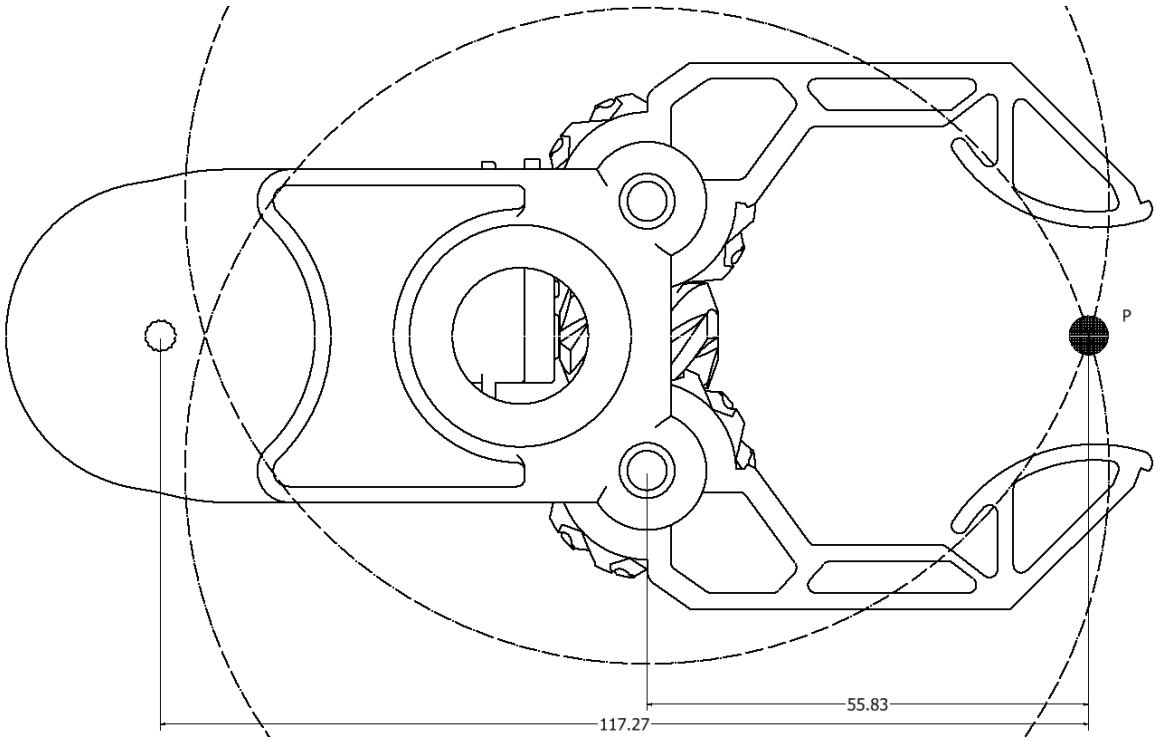
Obrázek 4.12 – Poměrové zatížení

Koncový efektor Konstrukci koncového efektoru jsem upravil na jednodušší. Pro pohyb čelistí používám šnekový převod. Servopohon, který je umístěný v gripperu,

ES08MA má krouticí moment 0,176 Nm, v programu Autodesk Inventor, ve kterém jsem návrh šnekového převodu dělal, bylo vypočten krouticí moment na šnekovém kole je $M = 0,389Nm$. Vzdálenost k bodu P je $r = 54,63\ mm$. Jednoduchým výpočtem $F = \frac{M}{r} = 7,07N$. Přibližná síla, kterou koncový efektor je schopný uchopit předmět je tedy 7 N. Ze znalosti statiky, aby předmět neklouzal při uchopení musí být síla $F > \frac{m*g}{\mu}$, kde μ je součinitel adheze, g je gravitační zrychlení a m je hmotnost předmětu. Dle testů v článku [44] největší koeficient tření s hodnotou 0,69 je z materiálu pro 3D tiskárny s technologií FDM (Fused Deposition Modeling) NinjaFlex. Tento materiál je však velmi obtížně tisknutelný a nejlepší design, který měl nejvyšší koeficient tření byl hladký povrch. Při zvedání obyčejných předmětů je výhodné mít měkký materiál na koncovém efektoru. Tímto materiál, který se přizpůsobí tvaru zvedaného předmětu může koeficient mít vyšší hodnotu. Tvar koncového efektoru je zaoblený z důvodu, že při zvedání větších předmětů může vzniknout velký úhel mezi předmětem a čelistí. Zaoblením dospěji ke snížení styčné plochy, ale působení normálové síly bude stále přibližně stejně velké. Tlustší vrstvou měkkého materiálu se při uchopení zvýší styčná plocha a je tímto možná kompenzace ztráty velikosti styčné plochy. Při zvedání předmětu je nutný i koeficient bezpečnosti, aby při pohybu předmět nevyklouzl. Největší přetížení je při pohybu směrem vzhůru, tedy proti směru gravitačního zrychlení. Z rovnice (4.2), jsem si vyjádřil hmotnost. Po dosazení hodnot $\mu = 0,69, F = 7\ N, \eta = 1, a = 0ms^{-2}$, kde μ je koeficient adheze, F je síla, η je koeficient bezpečnosti a a je zrychlení, mi vyšlo, že staticky udrží 0,98 Kg se silou 7 N, z toho důvodu pro manipulaci je tato síla s břemenem o 1 kg nereálná, ačkoliv dimenzování ramene bylo dostatečné. Konstrukční substitucí k pohonu ES08MA je ES08MD, který má krouticí moment 0,26 Nm. Po opětovném dosazení nových hodnot, $\eta = 1,5, \mu = 0,6, F = 11\ N$ a zrychlení v kolmém směru při zvedání předmětu $a = 2\ ms^{-2}$ vychází 0,75 kg. Hodnotu μ jsem snížil záměrně, neboť výsledná hodnota se jeví realističtěji. Vzhledem k tomu, že skutečný koeficient tření je neznámý, pak tyto výpočty jsou pouze orientační.

Jelikož má servopohon pouze rozsah pohybu 180° a potřebný rozsah pohybu čelistí je menší 90°, pak potřebný převodový poměr je 2:1. Z toho důvodu má šnek 4 závity.

$$2 * \mu * F = \eta * m * (g + a) \tag{4.2}$$



Obrázek 4.13 – Druhý návrh koncového efektoru

čelisti a šnek jsem vytisknul nejprve z materiálu PLA. Po vytisknutí jsem vyzkoušel, jak převod bude fungovat a zjistil jsem, že je příliš velké tření mezi zuby převodu. Jelikož vyhlazování povrchu PLA, jak jsem již zmínil, je náročné, tak z toho důvodu jsem vytiskl převod z materiálu ABS a vyhladil působením acetonových výparů. Vytvořil jsem si kelímek z alobalu, do kterého jsem vložil vytisknuté součástky. Do nádoby od barvy jsem nalil velmi malé množství acetonu a vložil jsem do něj kelímek z alobalu a uzavřel víkem. Po přibližně patnácti minutách jsem neviděl téměř žádný rozdíl. Po čtyrech hodinách to již bylo vyhlazené. Na obrázku 4.14 je vidět příliš velké rozpuštění čelisti, jelikož za tuto plochu to bylo položené v alobalovém kelímku.

Obrázek 4.14 –
Vytisknutý šnek (vlevo) a čelist (vpravo) po aplikování lázně acetonových výparů

4.3 Řídicí architektura

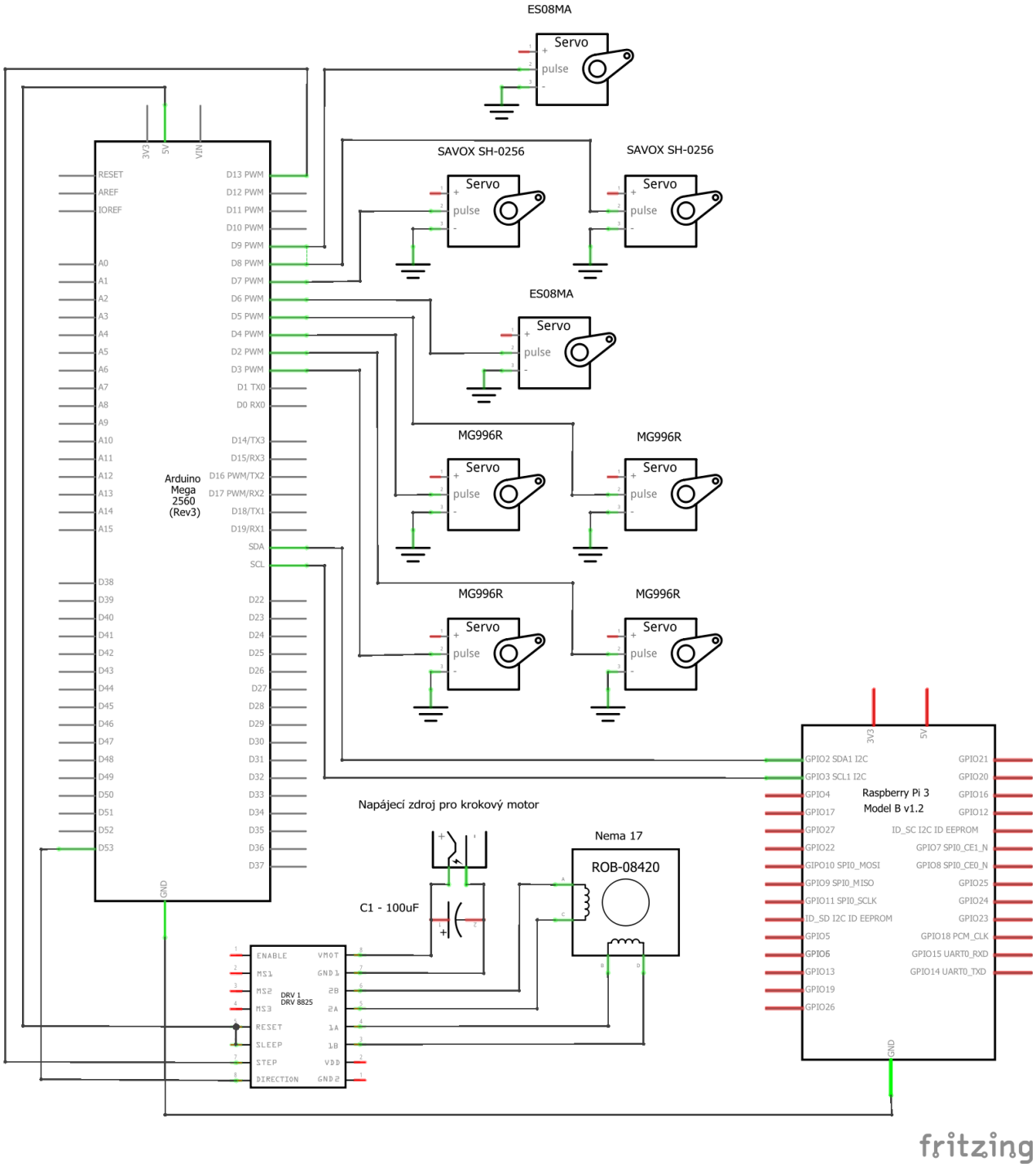
Z kapitoly 3 jsem si vybral Jetson Nano jako řídicí desku. Vycházela jako levná varianta s velkým výkonem bez nutnosti dokupovat další komponenty, jako například TPU procesor. Deska zvládne zpracovávat veškerá komplexnější data, jako jsou informace z kamery, inference neuronových sítí aj. Jetson Nano poté bude posílat instrukce přes I^2C do ovládací desky robota, kterým je Arduino. Servopohony jsou ovládané přes PWM (Pulse Width Modulation). Celkový počet servopohonů, kterými robot disponuje, je osm. Jelikož Arduino Uno má 6 PWM výstupů, musel jsem použít větší desku - Arduino Mega 2560, která má 15 PWM výstupů [45] a jeden krokový motor v základně robota, který je též ovládaný přes PWM.

V programu Fritzing jsem nakreslil schéma zapojení (4.15) [3]. Jetson Nano jako ovládací deska v programu Fritzing nebyla k dispozici. Napájecí napětí pro serva je 6 V, které jsem do schématu nekreslil z důvodu přehlednosti.

Komunikace I^2C funguje na bázi Master-Slave. I/O (Vstupní/Výstupní) piny Arduina pracují na napětí 5 V, zatímco Jetson Nano na 3,3 V. v případě zapojení Master (Jetson) - Slave (Arduino)[4], je možné je zapojit přímo. Avšak je potřeba obezřetnosti při zapojování, aby všechny pull-up rezistory byly připojené na referenční napětí 3,3 V. Jakýkoliv pull-up rezistor zapojený na 5 v může poničit zařízení s operačním napětím 3,3 V. Arduino má výstupy s otevřeným kolektorem a tudíž veškeré logické úrovně na I^2C lince budou nastavené na napětí desky Jetson Nano. Při zapojení v opačném případě by bylo nutné použít obousměrný převodník logické úrovně [47]. Data, která se budou přes sériovou linku posílat, budou úhly natočení servopohonů, které se vypočtou z inverzní kinematiky nebo vypočítají z rozpoznané postury snímané ruky člověka.

[3] Na schématu je použitý Raspberry Pi, ale zdroje [29] a [46] uvádí stejné rozložení I/O pinů u Jetson Nano a Raspberry Pi, tudíž substitutuovatelné

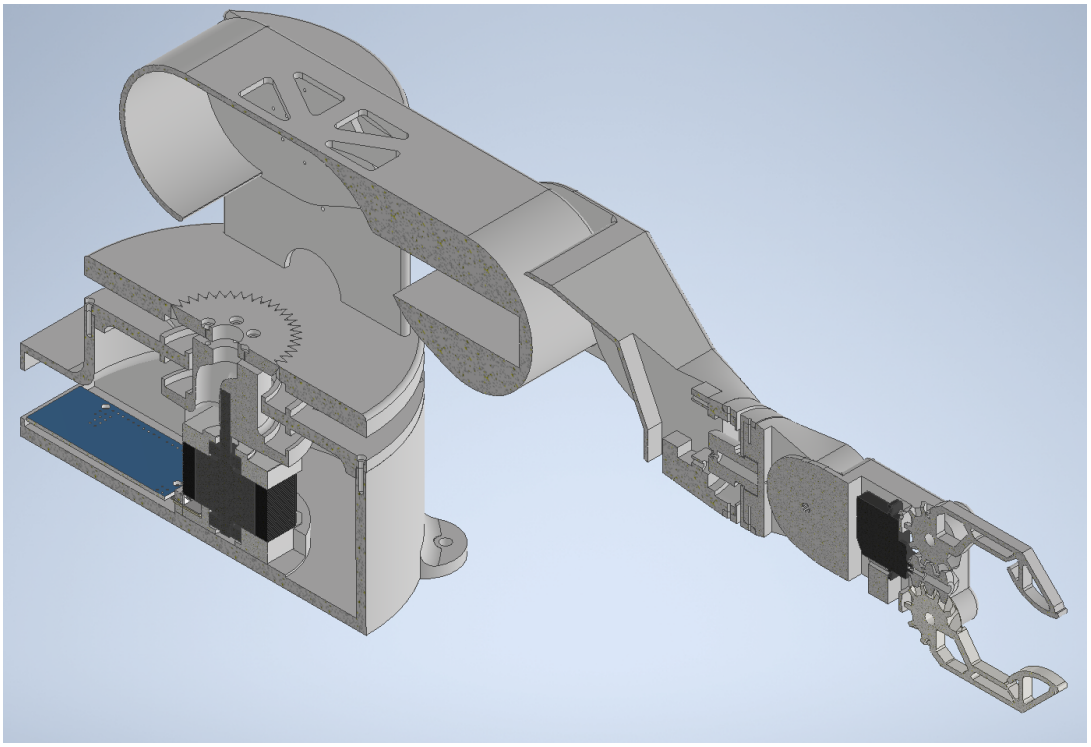
[4] Toto zapojení je chtěným případem zapojení v mé práci



Obrázek 4.15 – Schéma zapojení řídicí architektury

Průběžné zhodnocení výsledků

V sekci 4.1 jsem navrhl strukturu mechanismu pro manipulaci objektů s ohledem na ovládání člověkem za pomocí strojového vidění. v sekci 4.2 jsem v programu Autodesk Inventor zkonstruoval jednotlivé díly s ohledem na funkčnost konstrukce. Celá konstrukce se servopohony dokáže uzvednout přibližně 1,2 kg, ale síla v čelistech není dostatečně velká, aby uzvedla takto velkou zátěž, pokud by vazba mezi zátěží a čelistí byla obecná a záleželo by na smykovém tření. v kapitole 4.3 jsem navrhl, že jedna část robota - senzomotorická bude ovládat robota a druhá část, Jetson Nano bude posílat informace jak se mají servopohony natočit.



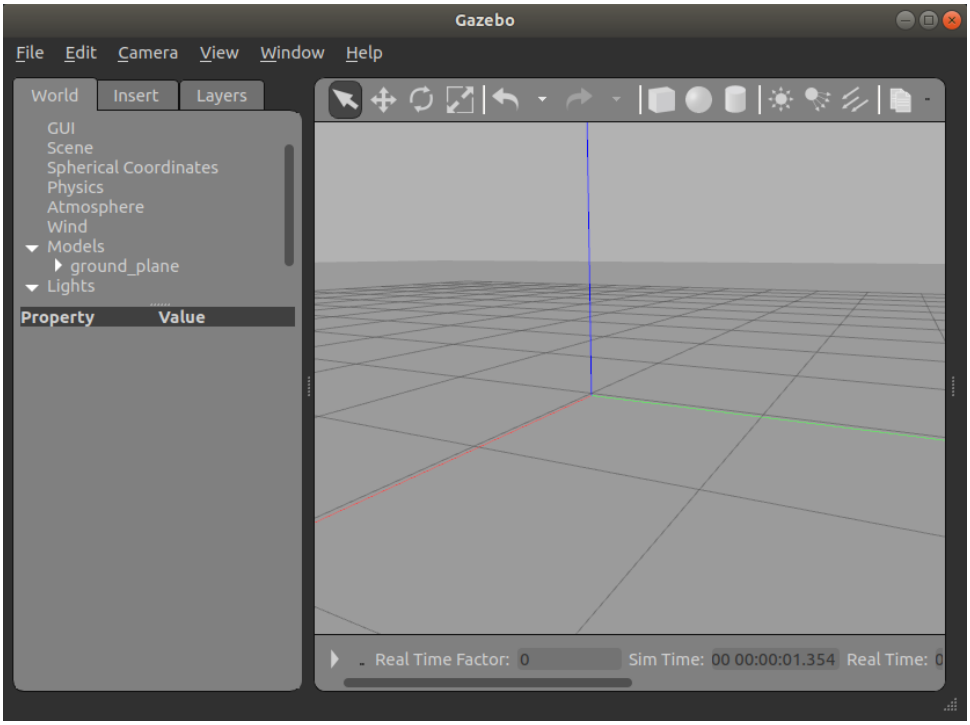
Obrázek 4.16 – Manipulační zařízení 3D pohled v řezu

4.4 Simulace

Simulační programy jsem si vybral ty, které jsou open source. Jelikož simulační programy obsahují podporu fyzikálního enginu, nebyla dynamická část v této práci řešena a ponechal jsem jí simulačnímu

4.4.1 Gazebo

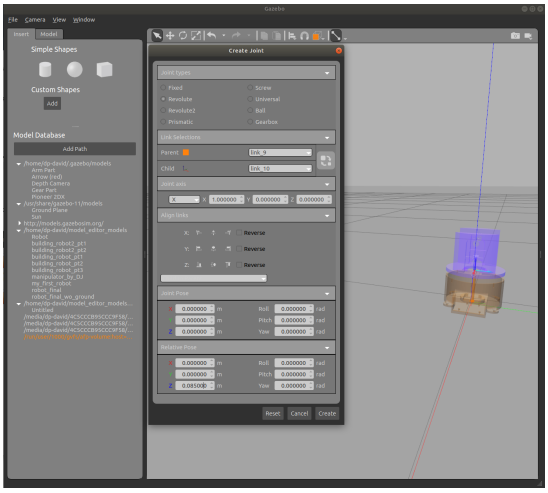
Gazebo lze používat s ROS nebo samostatně za pomocí pluginů, které to dovolují, ale je přednostně používán s ROS. Programování přes API softwaru se píše v jazyce C++.



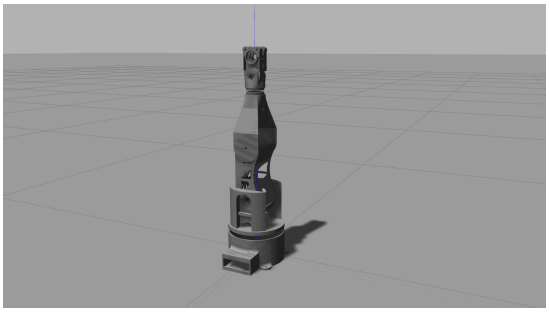
Obrázek 4.17 – Simulační prostředí Gazebo

Stavba robota

Stavba robota v Gazebu byla velmi jednoduchá. za pomoci klávesové zkraty ctlr+M nebo Edit → Model Editor na hlavní liště, jsem se dostal do rozhraní, ve kterém se robot sestavuje, definují se vazby, upravuje se hmotnost, těžiště a jiné fyzikální vlastnosti. Jako první jsem zvolil levém panelu tlačítkem Custom Shapes a importoval geometrii prvních dvou součástek, které jsem následně spojil rotační vazbou, jak je vidět na obrázku 4.18. Tento proces jsem opakoval, dokud robot nebyl sestaven celý (4.19) podle návrhu v části 4.2.



Obrázek 4.18 – Tvroba vazby v Gazebo simulátoru

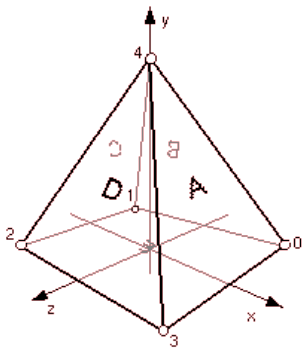


Obrázek 4.19 – Sestavený robot v simulačním prostředí Gazebo

Po nějaké době práce a pokusů propojit Gazebo s ROS společně s balíčkem OpenPose pro odhad postury, jsem narazil na problémy se softwarem, které jsem nedokázal překonat. Balíček OpenPose mi fungoval s ovladačem pro grafickou kartu s verzí 410 s potřebnými knihovnami CUDA 10.1 cuDNN ve verzi 7.6. Na operačním systému Linux jsem měl zapnuté automatické aktualizace operačního systému a ovladače pro grafickou kartu se tímto aktualizovaly a OpenPose mi již nefungoval z důvodů nekompatibility všech prvků [48]. Z těchto zmíněných důvodů jsem se rozhodl pro druhý simulační software Webots.

4.4.2 Webots

Oproti Gazebu se mi s Webots pracovalo mnohem lépe. Základním prvkem Webots jsou uzly, které mají názvy a jsou do nich zapsané parametry. Prvním objektem při spuštění nového projektu je WorldInfo, ve kterém jsou veškerá nastavení simulace. Mezi základní nastavení patří gravitační zrychlení, počet vykreslovaných snímků za sekundu, nebo časová délka kroku simulace. Zajímavější jsou například nastavení pro GPS souřadnice nebo ukazatel severu. Zobrazování 3D objektů je ve Webots přes uzel Shape, který obsahuje vlastnost geometrie. Do geometrie mohou být zapsány základní prvky jako je krychle, kužel, válec, koule, ale i jiné formáty pro zobrazování 3D grafiky, jako indexedFaceSet, indexedLineSet, PointSet [49].

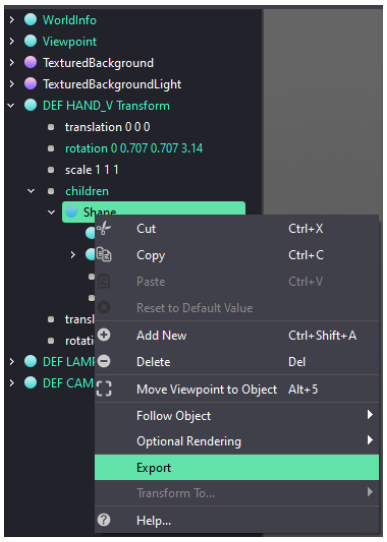


Obrázek 4.20 – Příklad indexovaného souboru ploch [49]

Webots obsahuje podporu pro ROS2, která však není podporována na operačním systému Windows. Po chvíli práce s ROS jsem dospěl k výsledku posílání dat do simulačního software, ale nastal neznámý problém. Uzel ve Webots tyto data číst nedokázal, ačkoliv uzel v ROS je přijímal. Z toho důvodu jsem po nějakém čase stráveným nad řešením tohoto problému zavrhl práci s ROS a program vyvíjel čistě v Pythonu.

Stavba robota

Do programu Webots nelze importovat žádný ze široce podporovaných standardních formátů pro CAD (STL, Obj, STEP, JT, apod.). Jediný formát, který lze přímo importovat je VRML97 (Virtual Reality Modeling Language) podle normy ISO/IEC 14772-1:1997 [50]. Importování geometrie do programu bylo složitější. Vydavatel programu vytvořil oddělený program jako zásuvný modul do 3D modelovacího programu Blender ve verzi 2.79b, ze kterého lze exportovat model do formátu, který je pro Webots nativní [51].



Obrázek 4.21 – Exportování tvaru

Pracovní postup pro importování geometrie do Webots je následovný:

1. Uložit model například do STL z Autodesk Inventoru (jednotky v metrech)
2. Otevřít model v Blenderu
3. Exportovat model do formátu nativní pro Webots (.WBT)
4. Vybrat uzel Shape v levé navigační liště a exportovat do .WBO (obr. 4.21)
5. Importovat uložený tvar.

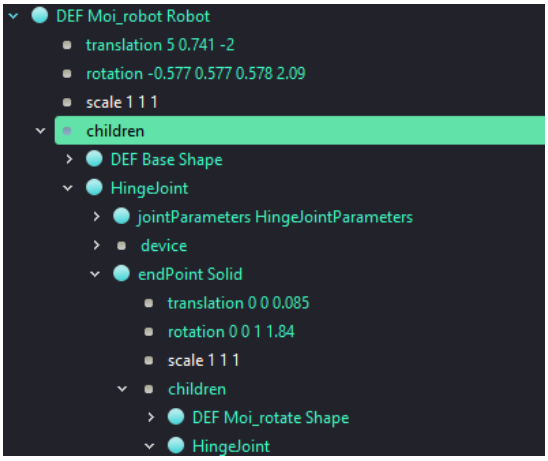
Pro stavbu robota existuje uzel Robot, který v sobě zahrnuje různé parametry jako ovladač (controller), ovládání přes dálku (remote controll) aj. Stavba robota se definuje vazbami mezi uzlem nadřazeným (parent) a podřazeným (children). Do podřazeného uzlu Robota jsem vložil tvar (Shape) základny, jako druhý podřazený uzel jsem vložil

uzel kloubový spoj (HingeJoint), což je uzel pro rotační vazbu. Do parametru device jsem vložil RotationalMotor a do parametru endPoint jsem vložil uzel Solid, do kterého lze vložit další tvar (shape) kinematické návaznosti. Aby bylo možné ovládat RotationalMotor je nutné k němu přiřadit i senzor. Zřetelněji je to uvedeno na obrázku 4.22. Tímto principem jsem vkládal objekty sériové kinematiky robota tak, jak jdou za sebou a je známo z kapitoly 4.2, včetně jejich posuvů na osové vzdálenosti, dokud nebyl robot v simulačním prostředí složený, jak je uvedeno na obrázku 4.23. Při každé konstrukci robota je doporučené dodržet souřadnicové systémy, které se v počátečním návrhu zavedou. Při nedodržení těchto souřadnicových systémů se může skládání robota v simulačním prostředí značně zkomplikovat.

Ovládací architektura v programu Webots

Uzel Robot ve Webots vlastní prvek ovladač (controller), na který je možné napojit program napsaný v C++, C, Pythonu, Matlabu dle jejich dokumentace. Z důvodu, že jde pouze o simulaci, jsem použil jazyky pro ovládání robota i pro posílání signálů z ovládací desky v jazyku Python. Všechny funkce, které jsou dostupné pro oživování robota v simulačním prostředí, neodpovídají reálně přenosnému kódu jedna ku jedné na ovládací desku Arduino, i kdyby byly napsané v jazyce C.

Ovladač pro Jetson Nano jsem nazval Jetson a ovladač pro Arduino Python_controller. Tyto dva ovladače spolu komunikují uvnitř programu Webots pomocí uzlů Vysílač (Emitter) a Přijímač (Receiver). Reálná kamera Intel RealSense je napojená na ovladač Jetson, kde je hlavní částí OpenPose (viz. 4.4.3), který zaznamenává klíčové body postury člověka. Po vypočítání úhlů z postury celé paže se tyto informace pošlou do Python_controlleru, kde se tyto úhly následně nastaví.



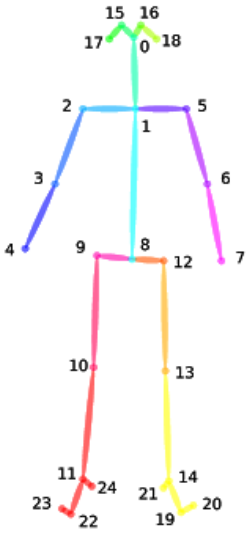
Obrázek 4.22 – Začátek struktury robota

Obrázek 4.23 – Sestava robota v simulačním prostředí Webots

Následně bylo nutné přejmenovat motory na názvy, kterými budou volány při nastavování úhlů. Je též nutné dát ukotvení pro pohyb v nastavení daného uzlu ve Webots, aby souřadnice rotace a zobrazované grafiky ladily.

4.4.3 Implementace OpenPose do webots

OpenPose je natrénovaná neuronová síť pro rozpoznání klíčových bodů na těle v místech kloubů, jako je zápěstí, loket, rameno a další. v novější verzi je i rozpoznání klíčových bodů pro klouby jednotlivých prstů na ruce. Na obrázku 4.24 je vidět kostra člověka z rozpoznaných klíčových bodů.



Obrázek 4.24 – Výsledek z OpenPose [52], s klíčovými body: 0 - Nos, 1 - Krk, 2 - Pravé rameno, 3 - Pravý loket, 4 - Pravé zápěstí, 5 - Levé rameno, 6 - Levý loket, 7 - Levé zápěstí, 8 - Střed boků, 9 - Pravý bok, 10 - Pravé koleno, 11 - Pravý kotník, 12 - Levý bok, 13 - Levé koleno, 14 - Levý kotník, 15 - Pravé oko, 16 - Levé oko, 17 - Pravé ucho, 18 - Levé ucho, 19 - Levý palec, 20 - Levý malíček, 21 - Levá pata, 22 - Pravý palec, 23 - Levý malíček, 24 - Pravá pata

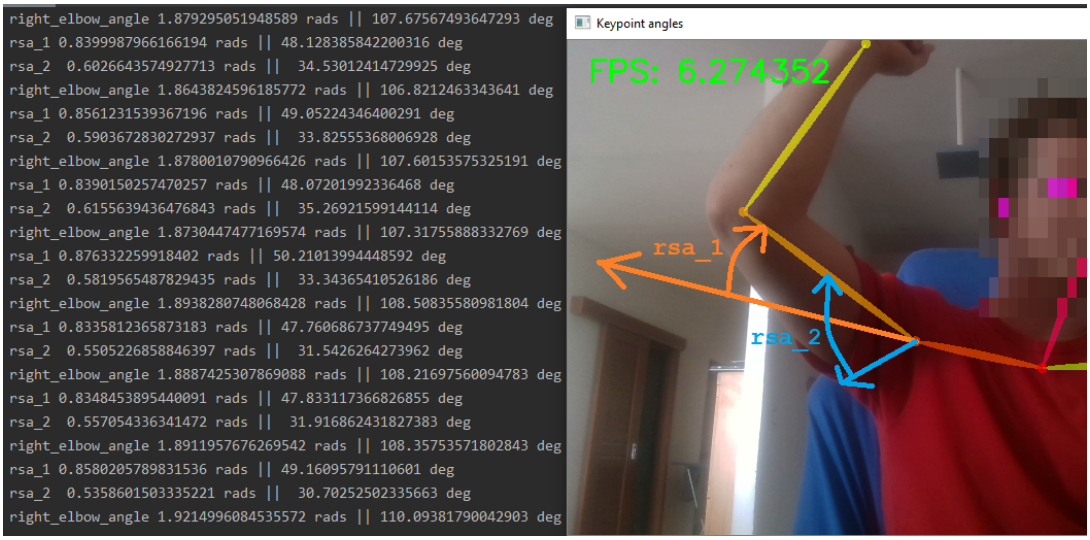
Z důvodů uvedených na konci části 4.4.1 jsem práci dělal na operačním systému Windows. Instaloval jsem dle postupu, který je uveden na GitHub v citaci [52].

Po úspěšném zprovoznění OpenPose jsem naprogramoval funkci, která umožní vzít bod z kamery a umístit ho do prostoru vůči souřadnicovému systému kamery. Hodnoty se však neshodovaly s realitou a bylo nutné kameru zkalibrovat dle postupu dostupného z dokumentace [53]. Po kalibraci již vše fungovalo tak, jak má. Dosáhl jsem toho, že všechny hodnoty z kamery se mi zobrazují v metrech a nikoliv výška x šířka v pixelech a hloubka v metrech.

Vytvořil jsem si funkci „get_component_points“, kterou volám vždy, když potřebuji sebrat jeden či více klíčových bodů, které mi OpenPose rozpozná. Abych si nemusel pamatovat dle obrázku 4.24 všechny indexy klíčových bodů, vytvořil jsem si vyhledávací tabulku, do které napíši jméno komponenty a vrátí mi jeho index.

Odměřování úhlu v lokti jsem vypočítal z trojúhelníku klíčových bodů ramene, lokte a zápěstí. Jelikož osy manipulačního zařízení jsou v těchto částech všechny v jedné rovině, nebylo za potřeba žádných další úprav.

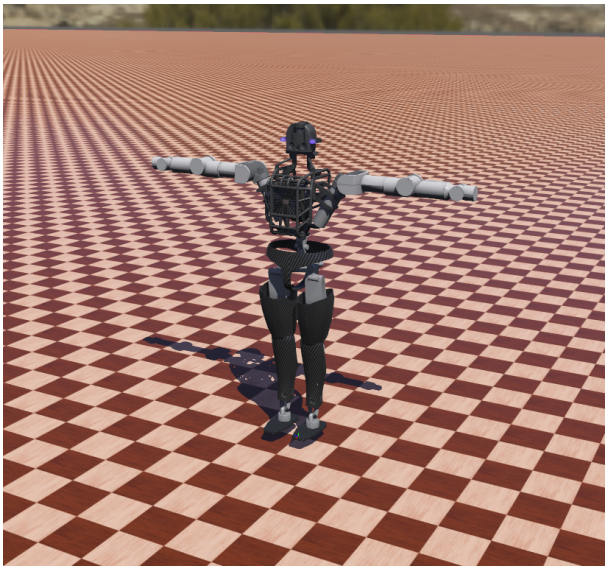
Pro správný odhad postury jsem potřeboval dva úhly. Jeden úhel jsem odměřil trojúhelníkem z klíčových bodů krk, rameno a loket tak, aby úhel ladil s určenou kinematikou manipulačního zařízení. Druhý úhel je odměřován z vektoru ramene, který jde do prostoru, dle obrázku 4.25.



Obrázek 4.25 – OpenPose výsledek -
rsa_1 - Úhel 1 pravého ramene; rsa_2 - Úhel 2 pravého ramene

Tyto úhly jsem posílal dle řídicí architektury do ovladače od robota. Problém, který nastal je ten, že model OpenPose si na grafické kartě alokuje pro načtení neuronové sítě přibližně 2,2 GB paměti, zatímco na svém stroji mám dostupnou paměť pouze 3 GB. A nejspíše z tohoto důvodu se simulace zastaví po druhé vteřině simulačního času a dostane se do stavu, kdy je možné pouze restartovat program, aby se mohl opět po druhé vteřině simulačního času zastavit.

Program Webots obsahuje mnoho ukázkových projektů, jeden z nich v je robot Atlas od firmy Boston Dynamics (viz. obr. 4.26). Atlas je humanoidní robot, který má kinematické řetězce stejné jako má člověk. Doufal jsem že problém, který jsem měl v případě svého manipulačního zařízení se ukáže pouze jako moje neznalost celého programu a sestavování robota. Po napasování několika zkušebních úhlu na robota Atlas z odměřování úhlů v OpenPose vyjde jako funkční řešení. Problém zde byl jiný, kdy simulační čas běžel, ale neobnovovaly se snímky ani postura robota. Po ukončení programu ovladače se robot nastavil do poslední známé polohy úhlů v OpenPose a vše v simulaci se od této doby již obnovuje.



Obrázek 4.26 – Atlas od firmy Boston Dynamics v simulačním prostředí Webots

4.4.4 Inverzní kinematika

Inverzní kinematika byla řešena balíčkem jménem IKpy [54]. Tento balíček dokáže pracovat s DH notací nebo se souborem pro universální popis robota (URDF - Universal Robot Description File). URDF je standardizovaný formát používaný v robotice. Slouží k popisu kinematiky i dynamiky robota jako celku. Avšak v něm nelze tvořit kinematické smyčky a nelze v něm specifikovat jiné vlastnosti, které nejsou patřičné k robotu. URDF používá Extensible Markup Language (XML) atributů pro popis kinematiky a jednotlivých vazeb [35]. Inverzní kinematika je v souboru „ik_robot_dp.py“.

Průběžné zhodnocení výsledků

V této části jsem aplikoval úspěšně odměřování úhlů v kloubech z modelu neuronové sítě OpenPose a vyzkoušel je přenést do simulačního prostředí Webots. Tento pokus byl úspěšný pouze z části. v příloze na videu ***vysledek_aplikace_openpose.mp4*** *je video jak se mi podařilo skrze kameru ovládat robota. Simulace mi vždy po 2,176 sekundách skončila a nemohl jsem přijít na problém, který tento výsledek způsoboval.*

4.5 Trénování neuronové sítě

Manipulační zařízení, které disponuje kamerovým systémem, může být efektivním nástrojem pro ovládání robota. Například při rozpoznání objektu a odhadnutí jeho polohy

je možné tento předmět uchopit a přemístit do potřebné polohy. Kamera s hloubkovým senzorem obsahuje informace jednotlivých bodů v prostoru. Detekované pozice objektu se teoreticky mohou za pomoci mračna bodů z kamery napasovat na 3D objekty v klasickém formátu STL, OBJ aj. Každý tento formát obsahuje informaci o lokálním souřadnicovém systému, z napasování mračna bodů a ze známosti celé montážní soustavy z CAD (Computer Aided Design) lze vyvodit požadovanou pozici a způsob montáže [5].

Rozpoznání objektu z obrazu je dnes standardní disciplína. Jedna její část je klasifikace objektu, která z inference neuronové sítě vyvodí, jaký objekt je na obraze. Jeho pozice, kde se na obraze nachází, není známá pro pouze pro klasifikaci. Existují jednoduché způsoby, jak zjistit pozici. Například inferovat natrénovanou neuronovou sítí ve fixní velikosti okna a iterovat po pixelech přes celý obrázek. Tento algoritmus je pomalý a pro reálně časové detekce objektů nepoužitelný, proto se používají architektury neuronových sítí, kde jejími vstupními daty jsou pozice objektů na obrázku.

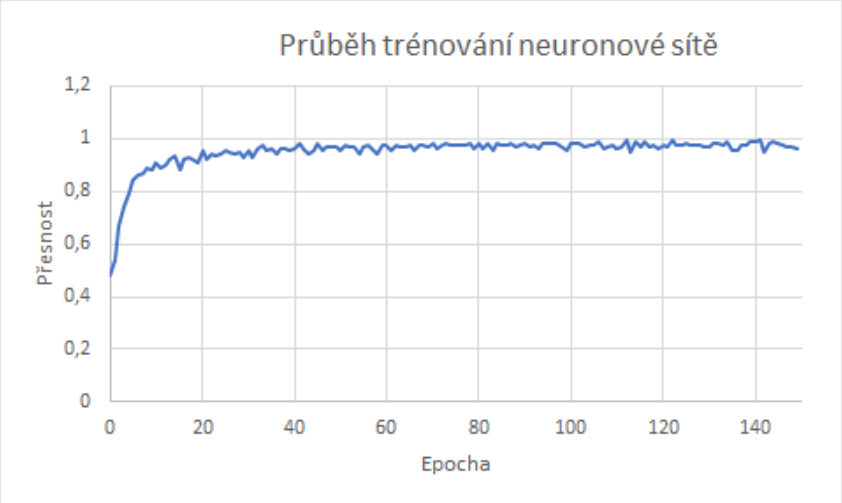
V případě trénování neuronové sítě na klasifikaci objektu je nutné říci, co se na obrázku nachází. Většinou bývá na obrázku pouze jeden předmět klasifikace. v případě detekce objektu je nutné ručně zadat souřadnice objektu, kde se na obrázku jaký objekt nachází. Tomuto procesu se říká anotace. Jelikož tento proces je časově náročný a nemám reálné objekty vytištěné, pokusím se natrénovat neuronovou síť. Nejdříve na klasifikaci, kde využiji rendery objektů z konstrukční části k dispozici. Poté se pokusím natrénovat neuronovou síť na klasifikaci.

4.5.1 Klasifikace objektu

Pro natrénování neuronové sítě, aby bylo možné rozpoznat objekty, jsem použil balíček s názvem Tensorflow. Jako zkušební objekty pro klasifikování jsem využil již zkonstruovaných částí z kapitoly 4.2. Jelikož se jedná o rozpoznávání objektu z obrázků, bylo nutné si vyrenderovat dataset, ze kterého se neuronová síť bude učit. k renderování jsem použil program Blender, ve kterém jsem si vyrenderoval 10 000 obrázků pro každý objekt. Již zmíněný balíček Tensorflow obsahuje funkci ImageDataGenerator, která namnoží obrázky, do kterých vnese různé deformace. Výsledkem tedy je 90 000 ob-

[5] Tato práce se touto problematikou nezabývá, pouze je motivací k řešení.

rázků rozdělených do pěti tříd a dále rozdělená na testovací a trénovací data. Nastavil jsem trénování na 150 epoch po 600 obrázcích na epochu. Z obrázku 4.27 je vidět, že je model dobře natrénovaný po uplynutí 40 až 60 epoch a další trénování bylo zbytečné, protože přesnost dále jen oscilovala. Z toho je možné odvodit, že dataset byl dostatečně velký pro natrénování menší neuronové sítě a stačilo by pouze přibližně 50 epoch*600 kroků = 30 000 obrázků v datasetu. s jistotou se nemůže určit, jak velký dataset bude potřeba k úspěšnému natrénování, ale většinou čím je dataset objemnější, tím je výsledek přesnější.



Obrázek 4.27 – Průběh trénování
Počet epoch (x), přesnost (y), kde 1 je velmi přesné

4.5.2 Blender

Blender je program pro tvorbu 3D objektů, renderů i animací. Obsahuje Python API, a můžou se psát programy k podpoře modelování a manipulaci s modely v Blenderu. Tuto vlastnost jsem využil pro renderování mnoha obrázků pro natrénování umělé neuronové sítě. Postup, který jsem použil je následovný:

1. Připravení osvětlení do scény včetně kamery mířící na objekt.
2. Importování 3D objektu pro renderování.
3. Nastavení pivotu[6] objektu do jeho těžiště a zarovnání na střed souřadnicového systému Blenderu.

[6]Pivot je bod kolem kterého se v prostoru provádí rotace objektu.

4. Spuštění programu níže uvnitř Blenderu.

```
import bpy
from math import *
from random import random
from random import gauss
import time

rot_objs = ["Hand"] # vybrání jména objektu který má být rotován

def render(filepath, viewport=False): # funkce pro renderovani
 ↪#viewport je renderování pro rychlý náhled
    bpy.context.scene.render.filepath = filepath
    if viewport:
        bpy.ops.render.render(write_still = True, use_viewport=True)
    else:
        bpy.ops.render.render(write_still = True)

def rotate_object_and_render(objects, iters, rot_axes=[0,0,0],
 ↪move_out=False, viewport=False):
    for x in range(iters):
        if move_out: # move_out je pohyb podel osy,
       # jev zmenseni ci zvetseni
            move_out = random.gauss(0, 0.05)
        if rot_axes[0]:
            rot_axes[0] += random()
        if rot_axes[1]:
            rot_axes[1] += random()
        if rot_axes[2]:
            rot_axes[2] += random()
        for obj in bpy.context.scene.objects:
            if obj.name in rot_objs:
                bpy.data.objects[f"{obj.name}"].rotation_euler =
 ↪rot_axes
                bpy.data.objects[f"{obj.name}"].location[1] = move_out
                print(f"Object rotated by {rot_axes}")

        render(f"/home/david/jetson_DP/Blender/Render/_hand/hand_{x}")

rotate_object_and_render(rot_objs, 10000, [1,1,1])
```

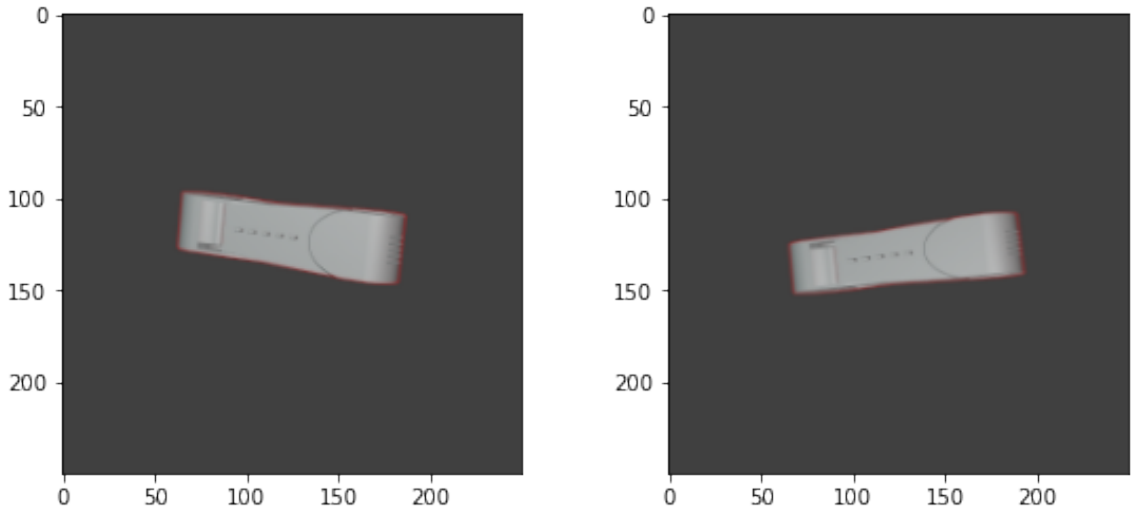
Do argumentů funkce rotate_objects_and_render se vkládají tři parametry. Prvním parametrem je list objektů, druhým parametrem je počet obrázků pro renderování a posledním parametrem je list jakých osách se má náhodně otáčet. Tento list je kombinací nul a jedniček, kde [1,0,1] se bude otáčet kolem osy x a z. Doporučuji si ručně nastavit

renderování za pomoci grafické karty, je-li k dispozici. Na procesoru mi trval jeden render přes 3 vteřiny, zatímco render na grafické kartě byl až stonásobně rychlejší. Při renderování jsem použil zvýraznění hran (outline), abych vykreslil okolo objektu ohraničení pro lepší kontrast mezi objektem a pozadím. Velikost renderovaného obrázku jsem nastavil na 250x250 pixelů.

4.5.3 Dataset

Dataset je rozdělený na trénovací data a testovací data. v každém z těchto adresářů je počet složek rovný počtu objektů, které trénuji. V mém případě 5 - ARM, `BASEROT`, FOREARM, HAND a WRIST. Z deseti tisíc vyrenderovaných obrázků pro jeden objekt jsem rozdělil do složek na 8000 jako trénovací sada a 2000 jako testovací sada.

Všechny obrázky vypadají podobně jako na obrázku 4.28.



Obrázek 4.28 – Náhled renderovaný objekt (vlevo) a stejný zdeformovaný objekt (vpravo)

K natrénování neuronové sítě jsem použil balíček s názvem Tensorflow. Tensorflow vlastní nástroje, které obohatí trénovací data o nekvality jako je rozostření, změna rozlišení, ořezání aj. Nástroj se jmenuje ImageDataGenerator.

```
image_generator = ImageDataGenerator(width_shift_range=0.1,
                                      height_shift_range=0.1,
                                      rescale=1/250,
                                      shear_range=0.2,
                                      zoom_range=.2,
```

```
                                    horizontal_flip=True,
                                    vertical_flip=True)
```

Tímto příkazem se obrázky různě natočí, zvětší či zmenší, převrátí i zdeformují v závislosti na vstupních parametrech a obohatí se dataset pro natrénování robustnější neuronové sítě.

Následuje sestavení samotného modelu, což je proces, kdy se vkládají jednotlivé typy vrstev neuronové sítě za sebou.

Pro trénování neuronové sítě jsem použil ztrátovou funkci Categorical Cross-Entropy (CCE) s optimizérem ADAM. CCE je spojení Softmax 4.3 aktivační funkce s křížovou entropií 4.4. Použití této funkce je z důvodu, že klasifikuji data do více kategorií

$$f(s)_i = \frac{e^{s_i}}{\sum_j^C e^{s_j}} \tag{4.3}$$

$$CE = -\sum_i^C t_i log(f(s)_i) \tag{4.4}$$

Optimizér ADAM je kombinací Adaptivního Gradient Descentu a Root Mean Square Propagation. Je používán ve většině případů, jelikož se ukazuje jako velmi efektivní nástroj při učení hlubokých neuronových sítí.

Shrnutím modelu je vidět tabulka 4.4, která ukazuje jednotlivé vrstvy, jejich skutečné rozměry a kolik parametrů každá vrstva obsahuje. Jako čtvrtý parametr ve sloupci „Output Shape“ je vidět počet filtrů, které jsem si nastavil při sestavování neuronové sítě. None ve sloupci Output Shape (první parametr) je batch size[7], který nemusí být konstantní a nemusí být specifikovaný.

Model se skládá ze čtyř vrstev konvolučních a poolingových. Vrstva flatten udělá z průběhu konvolucí a poolování vektor. Tento vektor obsahuje informace o vlastnostech obrázku, který byly zjištěné konvolucí a poolováním. Po tomto vektoru následují již vrstvy které slouží ke klasifikaci obrázku. Jelikož poslední vrstva před vrstvou flatten má tvar (13, 13, 64), tak výsledný vektor je o velikosti 10 816, což vzniklo vynásobe-

[7]Batch size je počet trénovacích dat použitých v jedné iteraci

Tabulka 4.4 – Vrstvy neuronových sítí

Model: "sequential"

Layer (type)	Output Shape	Parameters
conv2d (Conv2D)	(None, 248, 248, 32)	896
max_pooling2d (MaxPooling2D)	(None, 124, 124, 32)	0
conv2d_1 (Conv2D)	(None, 122, 122, 64)	18496
max_pooling2d_1 (MaxPooling2D)	(None, 61, 61, 64)	0
conv2d_2 (Conv2D)	(None, 59, 59, 128)	73856
max_pooling2d_2 (MaxPooling2D)	(None, 29, 29, 128)	0
conv2d_3 (Conv2D)	(None, 27, 27, 64)	73792
max_pooling2d_3 (MaxPooling2D)	(None, 13, 13, 64)	0
flatten (Flatten)	(None, 10816)	0
dense (Dense)	(None, 128)	1384576
activation (Activation)	(None, 128)	0
dropout (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 5)	645
activation_1 (Activation)	(None, 5)	0

Total params: 1,552,261
Trainable params: 1,552,261
Non-trainable params: 0

ním jednotlivých rozměrů. Tato vrstva je poté napojena na Dense vrstvu (propojení všech neuronů v jedné vrstvě se všemi neurony ve vrstvě druhé). Poté je aktivační funkcí ReLU[8]. Dropout vrstva dělá to, že předchází přeučení neuronové sítě vypínáním různých neuronů při trénování. Výsledkem přeučení je to, že by neuronová síť byla schopna správně klasifikovat pouze obrázky, na kterých byla trénována a testována. Dropout bylo nastaveno na 50 % , čímž se vypne každý druhý neuron v této vrstvě při trénování. Jako poslední je Dense vrstva, která je vrstvou výstupní do potřebné klasifikace. Jelikož mám 5 objektů, které bych chtěl rozpoznat, tak velikost poslední - výstupní vrstvy musí mít stejný rozměr. Poslední vrstva má sigmoidní[9] aktivační funkci. Všechny konvoluční vrstvy mají aktivační funkci ReLU.

Takto poskládaná neuronová síť má přes 1,5 milionů trénovatelných parametrů. Následuje samotné obohacení obrázků z vytvořeného konstruktoru image_generator. Funkcí "flow_from_directory" se dataset obohatí o zdeformované obrázky jako je vyobrazeno na obrázku 4.28. Je důležité mít target_size stejný jako všechny ostatní obrázky, neboť

[8]Rectified Linear Unit (ReLU) $f(x) = x^+$, kde x^+ jsou všechna kladná reálná čísla
[9]Sigmoidní aktivační funkce $g(x) = \frac{1}{1+e^x}$

první vrstva neuronové sítě očekává input_shape ve velikosti obrázku. Skutečný input_shape, se kterým počítá, je menší o pool_size, neboť se vytvoří rámeček v okolí, aby pooling neměl problémy s okraji.

```
batch_size = 16

training_image_generator =
 image_generator.flow_from_directory("robot_parts/train",
                                     target_size=(250,250),
                                     batch_size =batch_size,
                                     class_mode="categorical")

Output: Found 90000 images belonging to 5 classes.
```

```
test_image_generator =
    image_generator.flow_from_directory("robot_parts/test",
                                     target_size=(250, 250),
                                     batch_size = batch_size,
                                     class_mode="categorical")

Output: Found 10000 images belonging to 5 classes.
```

Podle složek se roztřídila data do kategorií a ke každé třídě se přiřadilo číslo třídy, do které objekt patří. Dále vidím, že jsem rozšířil dataset na 90 000 obrázků v trénovacím datasetu a na 10 000 obrázků v testovacím datasetu.

```
training_image_generator.class_indices
Output: {'ARM': 0, 'BASEROT': 1, 'FOREARM': 2, 'HAND': 3, 'WRIST': 4}
```

4.5.4 Trénování modelu

Trénování modelu na grafické kartě je příkazem `with tf.device(‘‘device_name’’)`, kde `device_name` je zjistěné příkazem `tf.test.gpu_device_name`. Do funkce fit jsou důležité parametry, jako je počet epoch, počet kroků na epochu a trénovací data společně s validačními daty.

Je lepší dávat `steps_per_epoch` mnohem menší než trénovací dataset a trénovat přes více epoch. Tím se získají i data o průběhu trénování skrze epochy příkazem (v mém případě results, kam výsledky ukládám) `results.history` - kde se mi vypíše slovník s hodnotami skrze epochy, které se může vykreslit do grafu.

Následně model uložím příkazem `model.save(„cesta/složky")`

```
with tf.device('/device:GPU:0'):
    results = model.fit(training_image_generator,
                        epochs=30,
                        steps_per_epoch=600,
                        validation_data=test_image_generator,
                        validation_steps=30,
                        callbacks=[cp_callback])

model.save("./models/model3")
```

4.5.5 Testování inference

Model lze načíst následujícím příkazem do proměnné

```
Model = tf.keras.models.load_model("./models/model3")
```

Načetl jsem obrázek ze složky `FOREARM`, jeden náhodně vybral a konvertoval na pole hodnot. Predikce bere tvar obrázku (1, dimenze_x, dimenze_y, pocet_kanalu). Obrázek se tedy musí rozšířit pomocí funkce `numpy expand_dims` pro získání stejného tvaru.

```
from tensorflow.keras.preprocessing import image
import numpy as np

img_to_predict = image.load_img("robot_parts/test/FOREARM/Forearm_8011.
 ↪png", target_size=(250,250))
img_to_predict
```

```
img_to_predict = image.img_to_array(img_to_predict)
img_to_predict = np.expand_dims(img_to_predict, axis=0)

Output: (1, 250, 250, 3)
```

Poté je doporučeno data znormalizovat. Funkcí `predict` získám pole, kde nejvyšší hodnota představuje míra určitosti rozpoznaného objektu. V mém případě je určitost 1, tedy na 100 % si je neuronová síť jistá, že na obrázku se nachází objekt na pozici 2 v poli.

Výše jsem se zmiňoval o rozřazování do tříd příkazem `training_image_generator.class_indices`, který vrátí {'ARM': 0, 'BASEROT': 1, 'FOREARM': 2, 'HAND': 3, 'WRIST': 4}. Tedy pozice 2 v poli má index `FOREARM`,

což je správně.

Další způsob je funkce `predict_classes()`, který vrátí index objektu, který má nejvyšší hodnotu - tedy 2, což je správně.

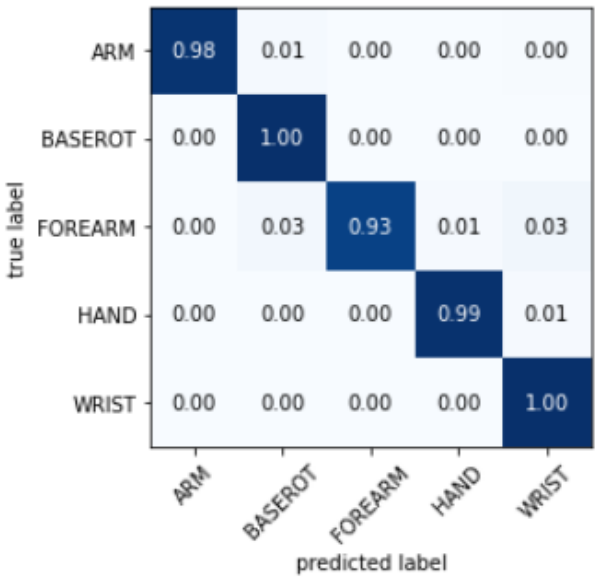
```
img_to_predict = img_to_predict/255 # normalizace
```

```
Model.predict(img_to_predict) # predikování obrázku


Output: array([[1.2949886e-17, 0.0000000e+00, 1.0000000e+00, 4.
 ↪5851006e-37, 1.7928691e-21]], dtype=float32)
```

```
Model.predict_classes(img_to_predict) # predikování třídy obrázku


Output: array([2], dtype=int64)
```

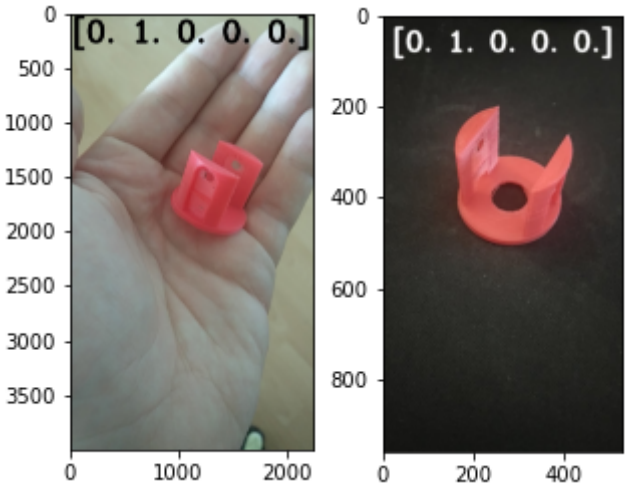
Způsob testování úspěšnosti rozpoznávání obrazu je confusion matrix. Confusion matrix (matice záměn), je matice, u které na svislé ose jsou hodnoty, které z trénovacích dat víme, že jsou správně a na vodorovné ose jsou hodnoty, které byly predikovány.

Vykreslím do grafu 4.29 výslednou matici záměn. Způsob interpretace dat je následující: v 98 % případů neuronová síť rozpoznala úspěšně třídu `ARM` a v 0,01 % případů si myslela, že je to `BASEROT`.



Obrázek 4.29 – Matice záměn

Jako posledním testem, zda byla neuronová síť natrénována dobře, je inference reálné fotografie. Pro tento test jsem si vybral část konstrukce manipulačního zařízení `BASEROT` a vytiskl ve zmenšené podobě na 3D tiskárně. Objekt ve zmenšené podobě obsahuje nedostatky jako jsou vruby, příliš tenké stěny a objekt je tisknutý červeným filamentem. Tyto fotky jsem načetl neuronovou sítí a vyhodnotil, na obrázku 4.30 jsou vidět vykreslené obrázky a text v horní části znamená pole jistoty, který objekt byl s jakou jistotou rozpoznaný. Objekt `BASEROT` je stále na druhém místě v poli a neuronová síť má jistotu 100 %, že objekty na obrázcích jsou `BASEROT`.



Obrázek 4.30 – Inference fotografií (textem v horní části obrázku je vektor klasifikace výsledků)

Těmito pokusy jsem si ověřil funkčnost, že lze z renderovaných objektů natrénovat neuronovou síť pro klasifikaci objektů. Ačkoliv byl objekt rozpoznán správně, je ještě nutné určit jeho pozici na obrázku - tedy detekce objektu. Jelikož existují architektury neuronových sítí pro detekci objektů, rozhodl jsem se je využít.

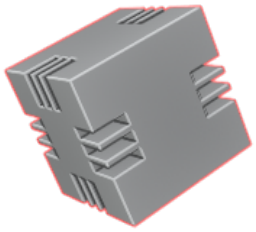
4.5.6 Detekce objektu

Detekce objektu je vyhodnocení, kde se na vstupních datech určitý objekt objevuje. Takových modelů je několik, například *Faster R-CNN*, *YOLO*. Jejich princip zde nebudu rozebírat a je možné si o nich přečíst na následujících odkazech [55] a [56]. Nejprve jsem vyzkoušel neuronovou síť *YOLO* verze 5 [57]. Tato verze neměla velkou úspěšnost při klasifikaci objektu, tak při její detekci. Jelikož je model *YOLO* brán jako velmi efektivní model s velkou přesností, tak jsem dlouho hledal důvod, proč mi neuronová síť nejde dobře natrénovat. Na fóru [58] jsem se poté dozvěděl, že tato verze je od jiného programátora, než jsou originální předešlé čtyři verze, poslední je tedy *YOLOv4* k datu psaní této práce [59].

Jelikož chci sledovat pohyb několika objektů, tak byly vytvořeny 3 kostky, které se liší velikostí a počtem drážek (viz 4.31, 4.32, 4.33).

Obrázek 4.31 – 10mm kostka

Obrázek 4.32 – 15mm kostka



Obrázek 4.33 – 20mm kostka

Připravil jsem si dataset podobným způsobem jako je uvedený v kapitole 4.5.2. Rozdílem oproti trénování neuronové sítě na klasifikaci objektu je ten, že k detekci se potřebují na každém obrázku označit objekty a uvést k nim popisek o který objekt se jedná. U datasetu z fotek se tato práce vykonává ručně za pomoci softwarů, které jsou pro to určené. v případě mého generovaného datasetu jsem použil při renderování outline,

který zvýraznil hranice celého objektu, pak označování objektů bylo jednoduchou prací pro ručně napsaný program v Pythonu.

Pro natrénování jsem využil online editoru Google Colab, u kterého je možnost použití jedné z výkonných grafických karet. První pokus trénování neuronové sítě nebyl velice úspěšný, a to z toho důvodu, že štítkování nebylo stoprocentně dobře, jelikož outline jsem neměl nastavený na plně červenou barvu, a tak byly rámečky občas posunuté. Jako další nastavení jsem upravil renderování transparentního pozadí. Čas trénování byl přibližně 6 hodin, což se jevilo jako dostatečné, ale výsledek byl takový, že neuronová síť nerozpoznala nic ani na testovacím obrázku. Zkusil jsem tedy natrénovat z reálných fotografií. Vytiskl jsem si na 3D tiskárně kostky a přidal k nim gripper. Jelikož jsem potřeboval mít více pozadí na každé fotce, tak jsem si z internetu stáhl různé fotografie, které jsem si promítl na monitor. Na tento monitor jsem následně položil vytisknuté předměty a fotil mobilním fotoaparátem (náhled na dataset 4.34).

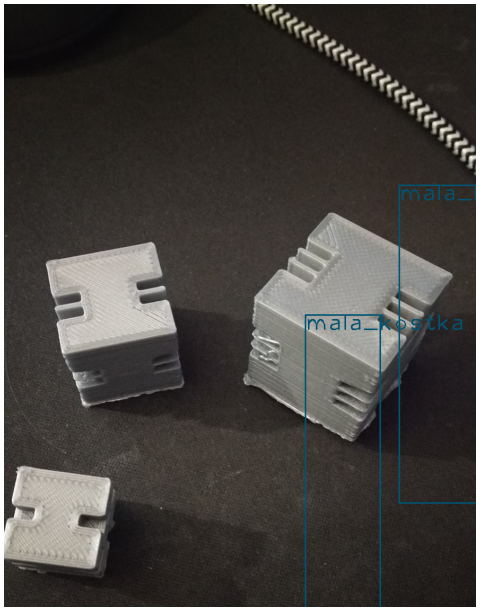
Obrázek 4.34 – Náhled na dataset

Na obrázku 4.35 je znázorněné štítkování objektů z obrázku v aplikaci LabelImg. Každý objekt má jiné barevné ohraničení dle popisu vpravo na obrázku.

Obrázek 4.35 – Náhled na štítkování v aplikaci LabelImg

Pro přípravu datasetu bylo nutné oštítkovat předměty. Pro tuto činnost jsem využil softwaru LabelImg [60], ve kterém se označí místo na fotce, na kterém je předmět umístěný a přiřadí mu správný název a následně se tyto informace uloží do požadovaného formátu neuronové sítě. Celkově jsem si připravil 160 fotek, o kterých jsem si myslel, že budou dostačující na vyzkoušení.

Pro natrénování jsem opět použil Google Colab, trénování běželo přes 5 hodin a poté se trénování dokončilo.



Obrázek 4.36 – Inference kostek

Obrázek 4.36 vykazuje výsledky, které jsou chybné. Při větším a robustnějším datasetu věřím, že by se neuronová síť natrénovala dostatečně. Jelikož byl proces celkově časově náročný a trvalo mi přibližně 4 hodiny tento dataset připravit, tak jsem se do dalšího tvoření obrázků nepouštěl.

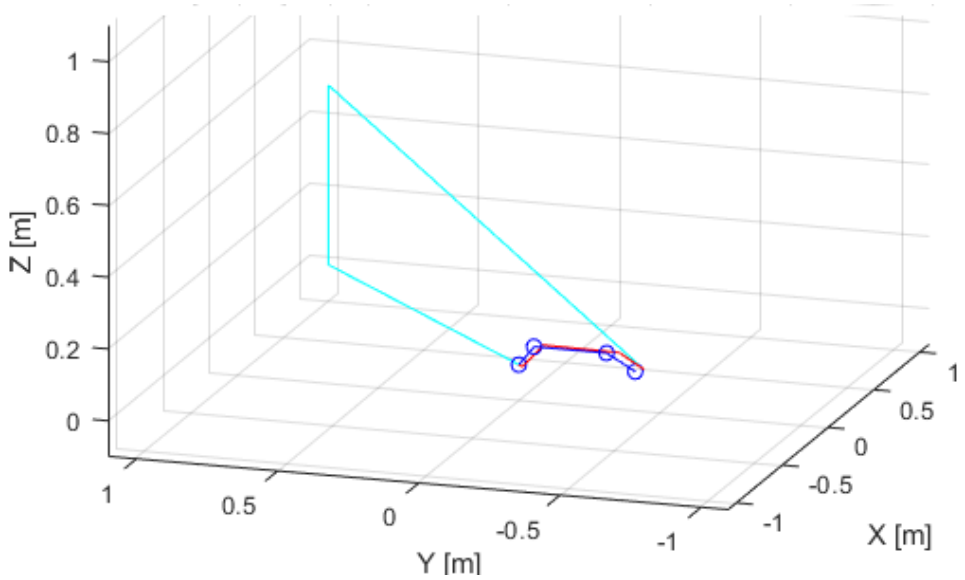
Průběžné zhodnocení výsledků

V této části jsem se pokusil natrénovat neuronovou síť pro klasifikaci objektu z obrázků s využitím renderů. k tomu jsem využil balíček TensorFlow a klasifikace objektu proběhla z úspěšně i z fotografie. Tímto jsem si ověřil, že z renderů se dá klasifikovat objekt a vyvodil jsem, že to lze i pro detekci objektu. Proto jsem si zvolil detektor YOLO a zkusil natrénovat neuronovou síť pro detekci objektu. Detekce objektu z nejasného důvodu nebyla úspěšná a nebyl rozpoznán ani jeden objekt. Využil jsem 3D tisku a objekty vytiskl, které jsem následně fotografoval vůči rozličnému pozadí. Avšak tento dataset byl malý, ale již jevil známky rozpoznání. Pro úspěšně natrénovaný detektor by bylo nutno více dat, které nejsem osobně schopen poskytnout.

4.6 Kalibrační algoritmus

Kalibrační alogritmus byl naprogramován pro kalibraci robota po jeho sestavení. Z důvodu, že v konečném důsledku byla prováděna pouze simulace, je uveden kalibrační algoritmus pro jeho budoucí použití.

Kalibrační algoritmus jsem vypracoval v nástroji MATLAB, dle kalibračního algoritmu uvedeného v sekci 3.2.3. Měřící přístroj pro kalibraci byl uvažován laser tracker a celkový počet kalibračních parametrů pro manipulační zařízení je 23. Významy jednotlivých kalibračních parametrů jsou uvedeny v tabulce 4.5. Kalibrační parametry jsem volil jako náhodné malé odchylky[10], které jsem přičítal v transformačních maticích. Vazbové rovnice jsou uvedené v (4.5). Je to série transformačních matic, kde kp_1 až kp_{23} jsou jednotlivé kalibrační parametry, S_1 až S_5 jsou natočení v kloubech manipulačního zařízení a d_1 až d_5 jsou konstrukční rozměry mezi vazbami, lt je změřené umístění laser trackeru na pozici x, y, z, V jsou naměřené hodnoty mezi laser trackerem do koutového odražeče v koncovém efektoru, které jsou již přepočítané do kartézského souřadnicového systému a vektor $\mathbf{r}_0 = [0\ 0\ 0\ 1]^T$.



Obrázek 4.37 – Zobrazení kostry robota a laser trackeru
Červená kostra - Robot s kalibračními odchylkami
Modrá kostra s kolečky - Robot bez kalibračních odchylek
Modré úsečky - Odměřování koncového efektoru laser trackerem

Hodnoty kalibračních parametrů byly vygenerovány náhodně a kalibrace proběhla přes 112 kalibračních poloh. Kalibrovatelnost robota dle rovnice (3.14) v kapitole 3.2.3, je přibližně $3 \cdot 10^6$, což je přijatelná kalibrovatelnost.

[10]Odchylky jsou větší než ty, které by mohly reálně nastat. Je to z důvodu viditelnosti při vykreslení odchylek jako je na obrázku 4.37

$$
\begin{aligned}
\mathbf{F} =& \mathbf{T}_x(kp_1)\mathbf{T}_y(kp_2)\mathbf{T}_{\varphi x}(kp_3)\mathbf{T}_{\varphi y}(kp_4)\mathbf{T}_{\varphi z}(S_1) \\
& \mathbf{T}_y(kp_5)\mathbf{T}_z(kp_6)\mathbf{T}_{\varphi y}(kp_7)\mathbf{T}_{\varphi z}(kp_8)\mathbf{T}_{\varphi x}(S_2) \\
& \mathbf{T}_y(kp_9)\mathbf{T}_z(d_2 + kp_{10})\mathbf{T}_{\varphi y}(kp_{11})\mathbf{T}_{\varphi z}kp_{12})\mathbf{T}_{\varphi x}(S_3) \\
& \mathbf{T}_x(kp_{13})\mathbf{T}_y(kp_{14})\mathbf{T}_{\varphi x}(kp_{15})\mathbf{T}_{\varphi y}(kp_{16})\mathbf{T}_{\varphi z}(S_4) \\
& \mathbf{T}_y(kp_{17})\mathbf{T}_z(d_3 + d_4 + kp_{18})\mathbf{T}_{\varphi z}(kp_{19})\mathbf{T}_{\varphi y}(kp_{20})\mathbf{T}_{\varphi x}(S_5) \\
& \mathbf{T}_x(kp_{21})\mathbf{T}_y(kp_{22})\mathbf{T}_z(d_5 + kp_{23})\mathbf{r}_0 - \\
& - \mathbf{T}_x(lt_x)\mathbf{T}_y(lt_y)\mathbf{T}_z(lt_z)\mathbf{T}_x(V_x - lt_x)\mathbf{T}_y(V_y - lt_y)\mathbf{T}_z(V_z - lt_z)\mathbf{r}_0
\end{aligned}
\tag{4.5}
$$

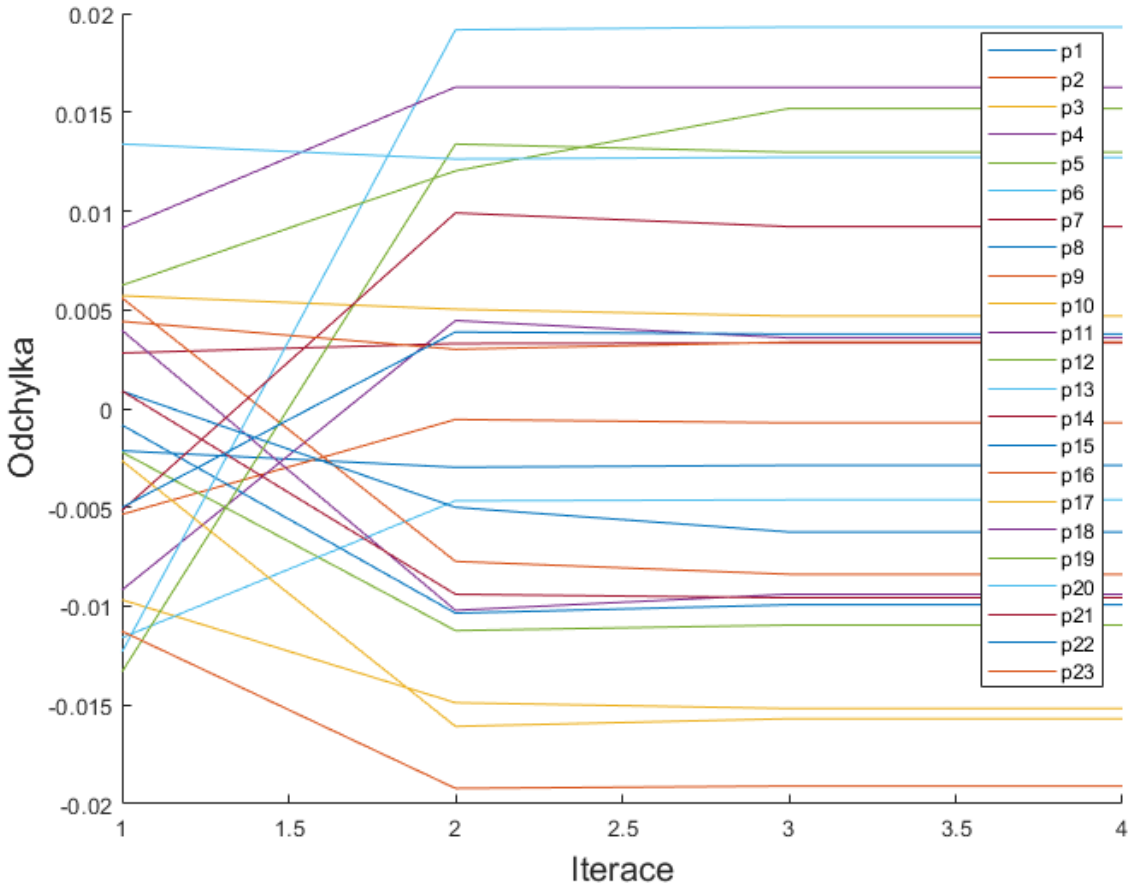
Tabulka 4.5 – Tabulka významů kalibračních parametrů
ofset [m], natočení [rad]

Kalibrační parametr	Význam
1	Ofset základny v ose X
2	Ofset základny v ose Y
3	Natočení v základně kolem osy X
4	Natočení v základně kolem osy Y
5	Ofset v ramením kloubu v ose Y
6	Ofset v ramením kloubu v ose Z
7	Odchylka natočení v ramenním kloubu kolem osy Y
8	Odchylka natočení v ramenním kloubu kolem osy Z
9	Ofset v loketním kloubu v ose Y
10	Ofset v loketním kloubu v ose Z
11	Odchylka natočení v loketním kloubu kolem osy Y
12	Odchylka natočení v loketním kloubu kolem osy Z
13	Ofset v ose rotace předloktí v ose X
14	Ofset v ose rotace předloktí v ose Y
15	Odchylka natočení v ose předloktí kolem osy X
16	Odchylka natočení v ose předloktí kolem osy Y
17	Ofset zápěstí v ose Y
18	Ofset zápěstí v ose Z
19	Odchylka natočení v zápěstí kolem osy Z
20	Odchylka natočení v zápěstí kolem osy Y
21	Posunutí do koncového efektoru držící koutový odražeč X
22	Posunutí do koncového efektoru držící koutový odražeč Y
23	Posunutí do koncového efektoru držící koutový odražeč Z

Tabulka 4.6 – Tabulka hodnot kalibračních parametrů

Číslo parametru	Hodnoty průběhu kalibrace (iterace)				Skutečné hodnoty
	1.	2.	3.	4.	
1	-0.0021	-0.0030	-0.0029	-0.0029	-0.0029
2	-0.0054	-0.0006	-0.0007	-0.0007	-0.0007
3	-0.0097	-0.0149	-0.0152	-0.0152	-0.0152
4	-0.0092	0.0045	0.0036	0.0036	0.0036
5	-0.0022	-0.0112	-0.0110	-0.0110	-0.0110
6	-0.0116	-0.0047	-0.0046	-0.0046	-0.0046
7	0.0028	0.0033	0.0033	0.0033	0.0033
8	-0.0008	-0.0104	-0.0099	-0.0099	-0.0099
9	0.0056	-0.0077	-0.0084	-0.0084	-0.0084
10	0.0057	0.0050	0.0047	0.0047	0.0047
11	0.0040	-0.0102	-0.0094	-0.0094	-0.0094
12	-0.0133	0.0134	0.0130	0.0130	0.0130
13	-0.0123	0.0192	0.0193	0.0193	0.0193
14	-0.0052	0.0099	0.0092	0.0092	0.0092
15	0.0009	-0.0050	-0.0062	-0.0062	-0.0062
16	0.0044	0.0030	0.0034	0.0034	0.0034
17	-0.0026	-0.0161	-0.0157	-0.0157	-0.0157
18	0.0091	0.0163	0.0163	0.0163	0.0163
19	0.0062	0.0120	0.0152	0.0152	0.0152
20	0.0134	0.0126	0.0127	0.0127	0.0127
21	0.0009	-0.0094	-0.0096	-0.0096	-0.0096
22	-0.0050	0.0039	0.0038	0.0038	0.0038
23	-0.0113	-0.0192	-0.0191	-0.0191	-0.0191

V tabulce 4.6 ve sloupci hodnot průběhu kalibrace jsou uvedené hodnoty pro každou iteraci kalibračního algoritmu. Tyto hodnoty jsou následně vykreslené do grafu 4.38. V sloupci skutečné hodnoty jsou hodnoty, které byly vygenerované náhodně před započetím odměřování kalibračních poloh a jsou tedy vnesenou odchylkou od ideálního stavu. Jelikož již třetí iterace průběhu kalibrace se shoduje s skutečnými hodnotami, pak kalibrace proběhla úspěšně a byly vypočteny kalibrační odchylky.



Obrázek 4.38 – Graf z tabulárních hodnot tabulky 4.6

Kapitola 5

Závěr

V této práci je zpracována rešerše manipulačních robotů, jejich struktury a řešení prostorových mechanismů. Dále jsem v rešerši uvedl použitelné vývojové desky a kalibrační algoritmus pro kalibraci robota. Tím byl v kapitole 3 splněn cíl 1. a 2.

Pro řízení pohybů robota bylo použito odměřování lidské ruky a inverzní kinematiky. v praktické části byla navržena struktura robota pro manipulaci, která je tvořena sériovou kinematikou s pěti stupni volnosti a je připodobněna lidské ruce. Stupně volnosti, které byly vynechány, jsou rotace ramenního kloubu v ose paže a zápěstí má pouze jeden stupeň volnosti. Tím byl v kapitole 4.1 splněn cíl 3.

v další části byla řešena konstrukce robotického manipulátoru se sériovou kinematikou a byly provedeny orientační výpočty pro dimenzování servopohonů. Konstrukce byla navržena tak, aby co nejvíce vodičů bylo skryto a po sestavení byl vzhled čistý a vodiče nevadily pohybu. Po návrhu konstrukce byl vypracován návrh řešení ovládání. Jako řídicí jednotky pro robota byly zvoleny dvě vývojové desky. První deskou je Jetson Nano, a druhou deskou je Arduino Mega 2560. Jetson Nano zpracovává informace z kamery a neuronové sítě, které následně vyhodnocuje a komunikuje s druhou deskou Arduino Mega 2560, která byla využita pro řízení pohonů. Jako kamera byla zvolena Intel RealSense D415, která obsahuje i hloubkový senzor. Ovládání gesty je v této práci skrze snímání postury člověka a odměřování klíčových bodů v kloubech, ze kterých byly vypočteny úhly, jejichž hodnoty jsou posílány do manipulačního zařízení. Pro rozpoznání postury člověka bylo využito již natrénovaného modelu OpenPose. Tím

byl v kapitole 4.2 splněn cíl 4.

Model OpenPose byl úspěšně implementován společně s vypočtením úhlů natočení v kloubech z dané postury. Dalším způsobem ovládání bylo rozpoznat objekt, který má manipulační zařízení uchopit. Pro rozpoznání objektu bylo využito neuronové sítě. Jelikož je nutné k trénování neuronové sítě mít data, tak bylo odzkoušeno trénování neuronové síť vlastních objektů za pomoci renderování snímků z programu Blender. Předpoklad byl takový, že půjde-li natrénovat neuronová síť pro klasifikaci objektu z renderovaných snímků, pak půjde natrénovat i pro jeho detekci. Pro klasifikaci byla natrénována neuronová síť a úspěšně odzkoušena klasifikace objektu z fotografie. Touto technikou bylo ověřeno, že lze klasifikovat objekty z 3D renderů natrénované neuronové sítě. U klasifikace objektu se může využít vyhledávacího algoritmu, u kterého je nevýhoda velmi malá rychlost vyhledání objektu. Z toho důvodu klasifikace objektu je nedostatečná pro aplikaci v praxi. Z možných detektorů byl vybrán model YOLO. Pro trénování tohoto modelu byly použity též renderované snímky, které byly anotovány programem napsaným v Pythonu, uvedeným na nosiči k této práci. Trénování bylo spuštěno přes šest hodin a z neznámého důvodu nepřineslo požadované výsledky, neboť nebyly rozpoznány ani trénovací data. Z toho důvodu byly objekty vytisknuty na 3D tiskárně a byl vytvořen menší dataset pro trénování neuronové sítě z fotografií. Pro robustnější natrénování byly použity různé fotografie jako pozadí na monitoru a objekty byly vůči tomuto pozadí fotografovány. Fotografie byly anotovány v aplikaci LabelImg, jelikož se musí označovat jednotlivé objekty ručně, je tento proces zdlouhavý. Touto cestou byla natrénována neuronová síť v čase přibližně pěti hodin a jevila známky rozpoznání. Rozpoznané objekty nebyly správně určené z důvodu malého datasetu. Pro trénování v obou případech bylo využito službu Google Colab, která nabízí zdarma výpočetní zdroje lepších grafických karet, než bylo k dispozici při této práci.

Detektory objektů obsahují o mnoho více parametrů, než neuronová síť pro klasifikaci objektů. Pro porovnání: sestrojený model v této práci pro klasifikaci obsahuje přes 1,5 milionů parametrů a použitý detektor YOLO třetí verze (YOLOv3), obsahuje 62 milionů parametrů [61]. Z toho důvodu je trénování náročnější na výpočetní výkon a i na čas.

Z důvodu nepředpokládaných událostí během roku 2020 bylo manipulační zařízení tisknuto na 3D tiskárně pouze částečně. Místo reálné konstrukce byl pokus o simulaci zařízení. Pro tento účel byl vybrán program Webots. Tento program obsahuje programovací API pro Python, ve kterém bylo využito modelu OpenPose. V průběhu práce nastaly technické problémy, které jsou softwarového i hardwarového charakteru. Po spuštění simulace s programem, ve kterém je implementován model OpenPose, se po pár sekundách zastavila a bylo nutno restartování celé simulace. Počítač, na kterém jsem práci dělal, je méně výkonný a OpenPose vyžaduje 2,2 GB grafické paměti. Při tvoření této práce byla k dispozici pouze grafická karta s 3 GB grafické paměti a spuštěním modelu se spotřebovaly téměř veškeré paměťové zdroje. OpenPose obsahuje i rozpoznání klíčových bodů dlaně a prstů. Rozpoznání prstů by sloužilo k rozeznání uchopení předmětu. Pro tuto část již nebyly k dispozici dostatečné hardwarové zdroje.

V simulačním programu Webots existují ukázkové projekty. Při pokusu o aplikaci OpenPose v jednom z těchto projektů bylo též problematické. V tomto případě se simulace nezastavila, nýbrž se jen neobnovovala a nebyly vidět jednotlivé kroky simulace. Po vypnutí ovladače se simulace obnovila a nadále se simulační kroky vypočítávaly v pořádku. Toto chování je odlišné od chování v prostředí mnou sestaveného robota, kdy se po dvou vteřinách všechno zastavilo a muselo se řešení restartovat. Řešení k nastalým problémům jsem hledal, ale bezúspěšně. Pro Jetson Nano je možné využít pro vývoj software a simulaci z dílny firmy NVIDIA vývojové prostředí Isaac. Problémem využití tohoto nástroje jsou požadavky na pracovní stanici [62], jejichž náklady jsou cenově v řádech desítek tisíc korun. Z toho důvodu jsem byl odkázán na softwary, které byly zdarma a mohl jsem je využít i na méně výkonném zařízení.

5.1 Možná vylepšení a odstranění problémů

1. Vyzkoušení jiných simulačních software pro robotiku.
2. Vyzkoušení jiných balíčků nebo frameworků pro rozpoznání postury člověka, které budou méně náročné na výpočetní techniku.
3. Vylepšení návrhu robotického manipulátoru pro plně antropomorfní kinematickou strukturu včetně dlaně a prstů.

4. Aplikování neuronových sítí s učitelem pro učení ovládání kinematických struktur, které jsou bližší lidské ruce.

5. Využití rozpoznání objektu a hloubkového senzoru pro montážní úlohu robotem.

Bibliografie

[1] S. Garrido-Jurado, R. Muñoz-Salinas, F. Madrid-Cuevas a M. Marín-Jiménez, "Automatic generation and detection of highly reliable fiducial markers under occlusion", *Pattern Recognition*, roč. 47, č. 6, s. 2280–2292, ISSN: 00313203. DOI: 10.1016/j.patcog.2014.01.005. WWW: https://linkinghub.elsevier.com/retrieve/pii/S0031320314000235 (cit. 13. 07. 2020).

[2] Z. Cao, G. Hidalgo, T. Simon, S.-E. Wei a Y. Sheikh, "OpenPose: Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields", *arXiv:1812.08008 [cs]*, květ. 2019, arXiv: 1812.08008. WWW: http://arxiv.org/abs/1812.08008 (cit. 26. 07. 2020).

[3] Doc. Ing. Vladimír ANDRLÍK, CSc. a kol., *Průmyslové roboty a manipulátory*. Ústav výrobních strojů a zařízení, 2012, sv. 2012.

[4] V. Brát, *Maticové metody v analýze a syntéze prostorových vázaných mechanických systémů*. ACADEMIA, sv. 1981.

[5] *Denavit-Hartenberg notation*. WWW: https://robotacademy.net.au/lesson/denavit-hartenberg-notation/ (cit. 14. 07. 2020).

[6] *Denavit–Hartenberg parameters*. WWW: https://en.wikipedia.org/w/index.php?title=Denavit%E2%80%93Hartenberg_parameters&oldid=963594857 (cit. 14. 07. 2020).

[7] *InverseKinematics*. WWW: http://motion.cs.illinois.edu/RoboticSystems/InverseKinematics.html (cit. 14. 07. 2020).

[8] P. Corke, *Task and Configuration space*. WWW: https://robotacademy.net.au/lesson/task-and-configuration-space/ (cit. 14. 07. 2020).

[9] Z. Šika, P. Beneš, V. Smutný a M. Valášek, *MECHANICAL CALIBRATION OF CLOPEMA ROBOTS*, Prezentace.

[10] V. Hamrle, "Kalibrovatelnost a její použití pro návrh paralelních kinematických struktur", Disertační práce, ČVUT v Praze, Praha.

[11] *RPi Distributions - eLinux.org*. WWW: https://elinux.org/RPi_Distributions (cit. 17. 09. 2019).

[12] Patrick, *15 best operating systems for Raspberry Pi (with pictures)*. WWW: https://raspberrytips.com/best-os-for-raspberry-pi/ (cit. 17. 09. 2019).

[13] Martins D. Okoi, *10 Operating Systems You Can Run on Raspberry Pi in 2019*. WWW: https://www.fossmint.com/operating-systems-for-raspberry-pi/ (cit. 17. 09. 2019).

[14] *Raspberry Pi 3: Specs, benchmarks & testing*. WWW: https://www.raspberrypi.org/magpi/raspberry-pi-3-specs-benchmarks/ (cit. 17. 09. 2019).

[15] D. Grieve, B. Cutler a J. Harmon, "Acknowledgements - Contributing Authors",

[16] A. Allan, *Benchmarking Edge Computing*. WWW: https://medium.com/@aallan/benchmarking-edge-computing-ce3f13942245 (cit. 17. 09. 2019).

[17] *Dev Board*. WWW: https://coral.withgoogle.com/products/dev-board/ (cit. 17. 09. 2019).

[18] ——, *Say "Hello" to Google Coral*. WWW: https://blog.hackster.io/say-hello-to-google-coral-cdbb49183864 (cit. 17. 09. 2019).

[19] *Buy the Latest Jetson Products*. WWW: https://developer.nvidia.com/buy-jetson (cit. 18. 09. 2019).

[20] *NVIDIA Jetson TX1 MODULE - 900-82180-0001-000*. WWW: https://www.siliconhighwaydirect.co.uk/product-p/900-82180-0001-000.htm (cit. 18. 09. 2019).

[21] *Jetson Nano: Deep Learning Inference Benchmarks*. WWW: https://developer.nvidia.com/embedded/jetson-nano-dl-inference-benchmarks (cit. 16. 07. 2020).

[22] *Tensor processing unit*. WWW: https://en.wikipedia.org/w/index.php?title=Tensor_processing_unit&oldid=913830528 (cit. 18. 09. 2019).

[23] Google, "NEXT TPU", Online prezentace, Published: Online prezentace, WWW: https://storage.googleapis.com/nexttpu/index.html (cit. 18. 09. 2019).

[24] *Cloud Tensor Processing Units (TPUs) \textbar Cloud TPU*. WWW: https://cloud.google.com/tpu/docs/tpus (cit. 18. 09. 2019).

[25] *Movidius*, čvc 2019. WWW: https://en.wikipedia.org/w/index.php?title=Movidius&oldid=905191979 (cit. 18. 09. 2019).

[26] ajolleyx, *Where to Buy \textbar Intel® Movidius™ Neural Compute Stick*. WWW: https://software.intel.com/en-us/neural-compute-stick/where-to-buy (cit. 18. 09. 2019).

[27] *Hands-on with the Google Coral USB Accelerator*. WWW: https://www.bouvet.no/bouvet-deler/hands-on-with-the-google-coral-usb-accelerator (cit. 18. 09. 2019).

[28] *GET A SPECIAL PRICE ON THE NEW NVIDIA JETSON AGX XAVIER MODULE*. WWW: https://www.siliconhighwaydirect.co.uk/product-p/900-82888-0060-000.htm (cit. 18. 09. 2019).

[29] *How to use PWM on Jetson Nano*. WWW: /blog/2020/05/27/configure-pwm-output-on-jetson-nano-m/ (cit. 16. 07. 2020).

[30] *Depth Camera D415*. WWW: https://www.intelrealsense.com/depth-camera-d415/ (cit. 13. 05. 2020).

[31] T. M. A. LLC), *Body tracking*. WWW: https://docs.microsoft.com/en-us/previous-versions/windows/kinect/dn799273(v=ieb.10) (cit. 13. 05. 2020).

[32] *OpenKinect*. WWW: https://openkinect.org/wiki/Main_Page (cit. 13. 05. 2020).

[33] M. Tenney, *Microsoft Kinect – Hardware*. WWW: http://gmv.cast.uark.edu/scanning/hardware/microsoft-kinect-resourceshardware/ (cit. 13. 05. 2020).

[34] *ROS.org About ROS*. WWW: https://www.ros.org/about-ros/ (cit. 03. 07. 2020).

[35] *Gazebo : Tutorial : URDF in Gazebo*. WWW: http://gazebosim.org/tutorials?tut=ros_urdf (cit. 09. 08. 2020).

[36] *Neuron – WikiSkripta*. WWW: https://www.wikiskripta.eu/w/Neuron (cit. 11. 07. 2020).

[37] N. S. Chauhan, *Introduction to Artificial Neural Networks(ANN)*. WWW: https://towardsdatascience.com/introduction-to-artificial-neural-networks-ann-1aea15775ef9 (cit. 11. 07. 2020).

[38] M. Nielsen, "Neural Networks and Deep Learning", s. 224,

[39] S. Saha, *A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way*. WWW: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53 (cit. 27. 07. 2020).

[40] *Image Kernels explained visually*. WWW: https://setosa.io/ev/image-kernels/ (cit. 27. 07. 2020).

[41] *MG996R Servo Motor Datasheet, Wiring Diagram & Features*. WWW: https://components101.com/motors/mg996r-servo-motor-datasheet (cit. 04. 07. 2020).

[42] *IO snímačů absolutního tlaku - dodávka do přístího dne \textbar RS Components*. WWW: https://cz.rs-online.com/web/c/polovodice/io-snimacu/io-snimacu-absolutniho-tlaku/ (cit. 04. 07. 2020).

[43] NatureWorks, *Solubility Study of Ingeo Biopolymer Grades in Common Organic Solvents*, čvc 2020. WWW: https://www.natureworksllc.com//media/Technical_Resources/Properties_Documents/PropertiesDocument_Solubility-Test-Results_pdf.pdf.

[44] J. M. G. Farstad, Netland a T. Welo, "Surface Friction of Rapidly Prototyped Wheels from 3D-Printed Thermoplastic Elastomers: An Experimental Study", *Procedia CIRP*, roč. 60, s. 247–252, 2017, ISSN: 22128271. DOI: 10.1016/j.procir.2017.02.009. WWW: https://linkinghub.elsevier.com/retrieve/pii/S2212827117300793 (cit. 03. 08. 2020).

[45] *Arduino Mega 2560 Rev3 Arduino Official Store*. WWW: https://store.arduino.cc/arduino-mega-2560-rev3 (cit. 16. 07. 2020).

[46] *GPIO - Raspberry Pi Documentation*. WWW: https://www.raspberrypi.org/documentation/usage/gpio/ (cit. 16. 07. 2020).

[47] D. Workshop, *I2C Between Arduino & Raspberry Pi*, zář. 2019. WWW: https://dronebotworkshop.com/i2c-arduino-raspberry-pi/ (cit. 16. 07. 2020).

[48] *cuDNN Support Matrix*, concept. WWW: https://docs.nvidia.com/deeplearning/sdk/cudnn-support-matrix/index.html (cit. 30. 07. 2020).

[49] *Webots documentation: Shape*. WWW: https://cyberbotics.com/doc/reference/shape (cit. 06. 07. 2020).

[50] *VRML97 Specification, ISO/IEC 14772-1:1997*. WWW: https://tecfa.unige.ch/guides/vrml/vrml97/spec/ (cit. 06. 07. 2020).

[51] *cyberbotics/blender-webots-exporter*. WWW: https://github.com/cyberbotics/blender-webots-exporter (cit. 06. 07. 2020).

[52] *CMU-Perceptual-Computing-Lab/openpose*. WWW: https://github.com/CMU-Perceptual-Computing-Lab/openpose (cit. 27. 07. 2020).

[53] *Dynamic Calibration User Guide for Intel® RealSense™ D400 Series*. WWW: https://www.intel.com/content/www/us/en/support/articles/000026723/emerging-technologies/intel-realsense-technology.html (cit. 27. 07. 2020).

[54] *ikpy: An inverse kinematics library aiming performance and modularity*. WWW: https://github.com/Phylliade/ikpy (cit. 09. 08. 2020).

[55] S. Ren, K. He, R. Girshick a J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks", *arXiv:1506.01497 [cs]*, led. 2016, arXiv: 1506.01497. WWW: http://arxiv.org/abs/1506.01497 (cit. 26. 07. 2020).

[56] J. Redmon, S. Divvala, R. Girshick a A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection", *arXiv:1506.02640 [cs]*, květ. 2016, arXiv: 1506.02640. WWW: http://arxiv.org/abs/1506.02640 (cit. 26. 07. 2020).

[57] *ultralytics/yolov5*. WWW: https://github.com/ultralytics/yolov5 (cit. 26. 07. 2020).

[58] *What is the latest version of YOLO? Is V5 a scam? · Issue #2198 · pjreddie/darknet*. WWW: https://github.com/pjreddie/darknet/issues/2198 (cit. 26. 07. 2020).

[59] A. Bochkovskiy, C.-Y. Wang a H.-Y. M. Liao, "YOLOv4: Optimal Speed and Accuracy of Object Detection", *arXiv:2004.10934 [cs, eess]*, arXiv: 2004.10934. WWW: http://arxiv.org/abs/2004.10934 (cit. 26. 07. 2020).

[60] darrenl, *tzutalin/labelImg*. WWW: https://github.com/tzutalin/labelImg (cit. 02. 08. 2020).

[61] L. Ong, *image slides*. WWW: https://lisaong.github.io/mldds-courseware/03_TextImage/image.slides.html (cit. 09. 08. 2020).

[62] *Setup — Isaac Sim 1.2 documentation*. WWW: https://docs.nvidia.com/isaac/isaac_sim/setup.html (cit. 03. 08. 2020).

Obsah datového nosiče

1. Složka Ostatní - Obsahuje krátké ukázkové video výsledku aplikace **vysledek_aplikace_openpose.mp4** a přílohy s výpočty aj.
2. Složka **Inventor** - Obsahuje všechny konstrukční soubory ve formátu nativním pro Autodesk Inventor. Z důvodu velikosti těchto souborů je uvedena pouze poslední konstrukční verze.
3. Složka **Webots** - Obsahuje aplikaci ve Webots. Programy jsou ve složce controllers. Pro otevření projektu je nutné načíst soubor **world.wbt** ve složce **worlds**.

   Aplikace je bez OpenPose - ten je nutné nainstalovat z návodu uvedený na stránce **https://github.com/CMU-Perceptual-Computing-Lab/openposeinstallation-reinstallation-and-uninstallation**, pro použití Pythonu se musí sestavit ve Visual Studiu Python zvlášť ve složce **openpose/python**. Celý OpenPose je poté nutné zkopírovat do složky **libraries**, nebo nastavit cesty pro OpenPose v programu **Jetson.py** na řádcích 60 a 77.

   Další nutností je instalace balíčků **Intel RealSense** (viz. **https://pypi.org/project/pyrealsense2/**).
4. Složka **Scripts** - Obsahuje všechny podpůrné programy pro práci, včetně kalibračního algoritmu.