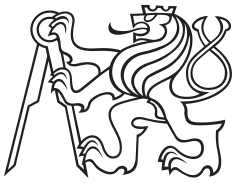
Bakalářská práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra kybernetiky

Ovládání humanoidního robota Tiago pomocí gest

Michal Procházka

Vedoucí: Mgr. Michal Vavrečka, Ph.D
Studijní program: Kybernetika a robotika
Květen 2023

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Procházka** Jméno: **Michal** Osobní číslo: **492048**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra kybernetiky**

Studijní program: **Kybernetika a robotika**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Ovládání humanoidního robota Tiago pomocí gest

Název bakalářské práce anglicky:

Gesture Operation of Humanoid Robot Tiago

Pokyny pro vypracování:

1. Vytvořte softwarový modul v prostředí ROS pro ovládání robota Tiago pomocí inverzní kinematiky a modul pro příjem obrazu z jeho kamer.
2. Otestujte ovládání teleoperace robota přes herní ovladač nebo klávesnici.
3. Vytvořte softwarový modul pro detekci člověka z kamerového obrazu a propojte jej s motorickým modulem (robot otáčí hlavou či celým tělem za detekovaným člověkem a snaží se jej umístit do středu zorného pole).
4. Vytvořte softwarový modul pro detekci končetin člověka a výpočet místa, kde ukazuje.
5. Otestujte přesnost pomocí úlohy, kde robot dojíždí do místa, kde člověk ukazuje.

Seznam doporučené literatury:

[1] TIAGO++ Handbook , PAL Robotics S.L., 2020.
[2] Chun, S., Park, S., & Chang, J. Y. (2023). Learnable human mesh triangulation for 3D human pose and shape estimation. In Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (pp. 2850-2859).

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Mgr. Michal Vavrečka, Ph.D. robotické vnímání CIIRC

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

Datum zadání bakalářské práce: **02.02.2023** Termín odevzdání bakalářské práce: **26.05.2023**

Platnost zadání bakalářské práce: **22.09.2024**

Mgr. Michal Vavrečka, Ph.D.	prof. Ing. Tomáš Svoboda, Ph.D.	prof. Mgr. Petr Páta, Ph.D.
podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

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

. | Datum převzetí zadání

Podpis studenta

Poděkování

Děkuji především Viktorii, mé holce z Unhoště, která po dobu psaní této práce byla ochotná poslouchat moje "bláboly o robotování" a přátelům Falešňákům za podporu, bez vás všech bych to rozhodně nedokázal.

Děkuji také své rodině za to, že mě během celého mého studia podporovala a věřila mi.

Dík patří i křečkovi Ibí, která mě čas od času kousla, aby mi dala najevo, že bych se měl zase věnovat práci a tolik neprokrastinovat.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze, 25. května 2023

Abstrakt

Tato práce se zabývá robotem TIAGo++, který je schopný svou kamerou sledovat člověka a na jeho příkaz dojet do místa na podlaze, do kterého člověk ukázal. Robot a jeho části důležité pro tuto práci jsou nejprve popsány, následuje návrh teoretického řešení jednotlivých podúkolů, jako jsou ovládání hlavy robota, rozpoznání člověka na obrazu z kamery, jeho zaměření do středu zorného pole robota, zpracování obrazu z kamery, způsob výpočtu směru ukazování ruky i finálního bodu a nakonec způsob pohybu robota po pracovní ploše.

Poté je popsána implementace teoretického řešení, způsoby komunikace s robotem, použité neuronové sítě pro zpracovávání obrazu a komunikace mezi nimi, vytvořené demo programy ověřující správnou funkci dílčích částí řešení a následně i hlavní pracovní smyčka, která se sestává z nalezení člověka, rozpoznání jeho gesta a přesunu do požadovaného místa. Na závěr je zhodnocena spolehlivost nalezení člověka robotem a také přesnost a opakovatelnost určení polohy dojezdu robota.

Klíčová slova: TIAGo++, ROS, detekce člověka, robot, gesta

Vedoucí: Mgr. Michal Vavrečka, Ph.D
CIIRC,
Jugoslávských partyzánů 1580/3
Praha 6

Abstract

This thesis describes a TIAGo++ robot capable of watching humans with its camera and moving to a place on the ground specified by a human when it is prompted to. At first, the robot and its parts important for this thesis are described, then follows the suggested theoretical solution of individual subtasks like controlling the movement of the head, recognizing the human in the image from the robot's camera, centering on the person in the image, processing of the image, means of computing the direction in which the human points and the location of a final point for the robot to move to and the way the robot moves around the workspace.

Then the implementation of the theoretical solution is described and is followed by means of communication with the robot, a description of neural networks used to process the images taken, and the communication between them. Demo programs created to verify the correct behavior of the program are mentioned and finally, the main loop of finding a human, keeping him in the robot's field of view, recognizing the gesture the person does, and moving to a place that was pointed to is examined. In the end, the reliability of detecting a person in the image and the accuracy and repeatability of the robot moving to a specified point are verified and discussed.

Keywords: TIAGo++, ROS, human detection, robot, gestures

Title translation: Gesture Operation of Humanoid Robot Tiago

Obsah

Obrázky

Tabulky

Kapitola 1

Úvod

Ve světě robotiky se čím dál tím více setkáváme s myšlenkou, že humanoidní roboti by mohli být dříve nebo později plnohodnotně začleněni do výrobních podniků a továren, kde by doplnili či přímo nahradili lidskou pracovní sílu. Tuto myšlenku razí například známý podnikatel Elon Musk, který se svým týmem představil prototyp humanoidního robota Optimus [HS22]. Musk nechává roboty trénovat vykonávání jednoduchých pracovních úkonů přímo v továrnách Tesly a plánuje humanoidní roboty prodávat za 20,000$, což je srovnatelné s cenou osobního automobilu. Během dokončování této práce dokonce přišel s vizí 10 miliard prodaných kusů. Dalším příkladem pokroku v oblasti ovládání humanoidních robotů je robot Atlas od společnosti Boston Dynamics [Dyn23b], která robota připravuje na vykonávání složitých pohybových sekvencí. Atlas je tak schopný překonat například překážkovou dráhu či manipulovat s předměty ve svém okolí [Dyn23a].

Tyto roboty a další jim podobné musí řešit otázku přesného pohybu v pracovním prostoru. Stejně tak je důležité, jak takového humanoidního robota vůbec ovládat. Je samozřejmě možné vyřešit ovládání pomocí teleoperace, avšak pak by takový robot nebyl schopný samostatného pohybu bez člověka. Jedno z příhodných řešení ovládání je robotu zadávat úkoly interaktivně, např. řečí nebo pohyby těla. Robot by mohl sledovat, co člověk dělá, či poslouchat, co říká, a na základě těchto vstupních podnětů vykonat danou činnost. Právě myšlenka na možnost ovládat robota pomocí gest či řeči byla hlavní inspirací pro tuto práci a realizace této myšlenky pak cílem této práce.

1.1 Cíl práce

Mým vytyčeným cílem je ovládat pohyb humanoidního robota TIAGo++ [RB22] pomocí vizuálních příkazů. Pohyb robota bude ovládaný v prostředí ROS [OR21] a program bude psaný v jazyce Python. Úkolem robota bude najít ve svém pracovním prostoru člověka pomocí své RGB-D kamery, a následně ho udržet ve svém zorném poli. Když robot vyhodnotí, že je člověk v centru jeho zorného pole a dále již se nehýbe, robot si určí pozici těla a místo, na které člověk ukazuje pravou rukou, spočte si kde se dané místo nachází na podlaze a v jeho souřadném systému a posléze se na určené místo přesune. Jakmile se robot přesune na požadované místo, otočí se do směru, kde se před pohybem nacházel člověk a pokusí se ho znovu detekovat. Důležitý výsledek této práce bude přesnost určení správného místa dojezdu z gest člověka a následná přesnost po několika opakováních tohoto úkonu.

Obrázek 1.1: Robot TIAGo++ [RB22]

1.2 Přínosy této práce

Hlavní přínosy této práce pro budoucí vývoj v oblasti humanoidní robotiky a pro další práce s robotem TIAGo++ jsou:

- Vytvoření uživatelského prostředí a funkcí pro možnost ovládání jednotlivých částí robota TIAGo++ pomocí jazyka Python.

- Využití veřejně dostupných a předtrénovaných neuronových sítí pro detekci gest člověka pomocí vlastní kamery robota.
- Implementace interaktivního ovládání pohybu robota v pracovním prostředí bez nutnosti znalosti programovacích jazyků, které pohyb zprostředkovávají.
- Vytvoření jednoduchých testovacích programů ověřujících správné fungování jednotlivých částí robota.

1.3 Další související práce

Tato práce představuje určité problémy a jejich řešení, které již byly tématem jiných prací různých autorů. Například způsoby ovládání různých částí robota TIAGo++ byly popsány Markem Jalůvkou v jeho diplomové práci [Jal21]. V práci řešil mimo jiné i způsob rozpoznání cíle robotem pomocí jeho vestavěné kamery. Ke správné kooperaci robota s člověkem musíme zajistit, že robot bude schopen rozpoznat pózu člověka a vyčíst z ní data pro své následující úkony.

Problému rozpoznání pózy člověka z obrazu se věnuje například práce [CPC22], ve které se její autoři zaměřují na možnost rozdělení úkolu rozpoznání člověka na dvě části. V první části se použije konvoluční neuronová síť k rozpoznání vrcholů sítě modelu člověka ze vstupního obrazu a následně se nalezeným vrcholům pokouší přizpůsobit model člověka nazývaný SMPL [LMR+15], který se používá k vizuální reprezentaci člověka ve virtuálním prostředí. Tato metoda rozpoznávání tvaru a rotace kloubů je pak významně výkonnější než předchozí postupy popisované ve zmiňované práci.

Myšlenka sledování záměrů člověka a přizpůsobení se jeho požadavkům již v tomto odvětví také není úplnou novinkou, například článek [ET10] se zabývá myšlenkou změny ovládání robota podle rozpoznaného úmyslu člověka pouze pomocí jeho pozorování. Robot se tak přepíná mezi čistě impedančním řízením s pevným referenčním bodem a interaktivním ovládáním dle úmyslů člověka.

Kapitola 2

Popis zkoumaného problému

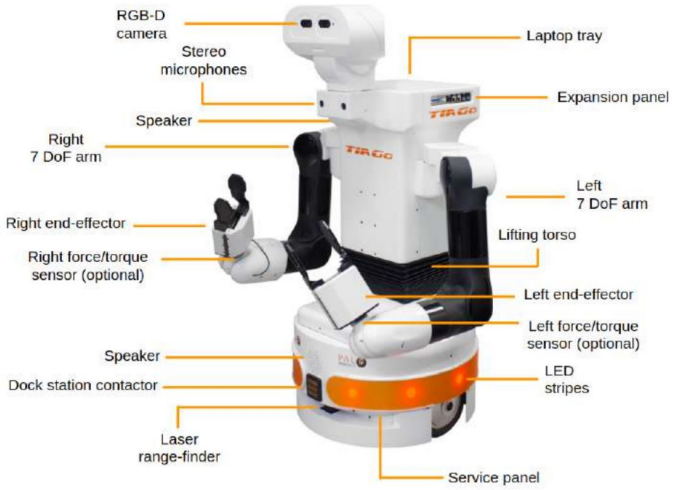
Klíčové problémy, které se tato práce pokusí vyřešit, se týkají způsobu ovládání jednotlivých kloubů a částí robota TIAGo++, provedení rozpoznání člověka a jeho pózy a následné vyřešení přesnosti opakovatelnosti prováděných pohybů po pracovním prostoru. Robota a jeho jednotlivé komponenty, které využijeme k naplnění cílů této práce, si nyní blíže představíme.

2.1 Robot TIAGo++

Pro tuto práci jsme si zvolili robota TIAGo++ [RB22], jelikož jsme s ním již měli zkušenosti z práce na bakalářském projektu. Zároveň je vybavený RGB-D kamerou, jež je k úkolu rozpoznání gest člověka klíčová. Na obrázku 2.1 je vidět podrobný popis všech jeho částí. Základní specifikace robota a některých jeho částí, které budeme v naší práci využívat jsou uvedeny v tabulce 2.1. Ty nejdůležitější z nich pro naši práci si více popíšeme v této kapitole

2.1.1 Hlava

Hlava robota TIAGo++ je umístěna na naklápěcím mechanismu s dvěma stupni volnosti, umožňující otáčení hlavy v horizontálním a vertikálním směru.



Obrázek 2.1: Robot TIAGo++ s popisem jeho částí [L.20]

Zároveň je vybavena stereo mikrofóny pro záznam zvuku, reproduktorem a také RGB-D kamerou. Vzhledem k tomu, že rozpoznání člověka kamerou v hlavě robota je klíčové pro správné dosažení požadované pozice robota, důležité parametry pohybového mechanismu jsem vypsal do tabulky 2.2. Důležitý parametr ve zmiňované tabulce je rychlost otáčení kloubů. Ve fázi hledání člověka v obraze bude důležité, jak dlouho se bude hlava robota otáčet, jelikož s tím bude spojená doba detekce člověka a schopnost robota udržet ho ve svém zorném poli. Při použití teleoperace k ovládání robota (viz sekce 4.4) jsem zjistil, že mezní úhel natočení hlavy v horizontálním směru na obě strany je

$$\theta_{max} \approx 70^{\circ} \tag{2.1}$$

Tento fakt je důležitý pro návrh algoritmu pokrytí pracovního prostoru zorným polem (viz sekce 3.3.1)

RGB-D kamera

Jelikož je kamera klíčovou částí pracovního cyklu naší práce, vypsali jsme si její, pro nás důležité parametry do tabulky 2.3. Kamera je také vybavena senzorem pro zachycení hloubkového snímku, ve kterém jsou obsaženy vzdálenosti jednotlivých pixelů v obraze. Tento senzor využijeme při hledání vzdálenosti člověka od robota v sekci 3.4.4. V tabulce je uveden například úhel horizontální šířky zorného pole. Znalosti tohoto úhlu pak můžeme využít například v sekci 3.2 při návrhu algoritmu pohybu hlavy pro pokrytí pracovního prostoru zorným polem kamery. V tabulce je uvedeno i použité rozlišení kamery, které budeme k rozpoznávání potřebovat. Hlava robota a umístění kamery na ní je ukázáno na obrázku 2.2.

Dimensions	Height	110 - 145 cm
	Weight	72 kg
	Base footprint	∅ 54 cm
Degrees of freedom	Mobile base	2
	Hey5 hand	19 (3 actuated)
	PAL gripper	2
Mobile base	Drive system	Differential
	Max speed	1 $m \cdot s^{-1}$
Sensors	Base	Laser range-finder
		Sonars
		IMU
	Torso	Stereo microfones
	Head	RGB-D camera

Tabulka 2.1: Hlavní specifikace robota TIAGo++ a jeho částí důležitých pro tuto práci [L.20]

Description	Reduction	Max speed [rpm]	Max torgue [Nm]	Absolute encoder
Pan motor	200:1	63	6	12 bits
Tilt motor	200:1	63	6	12 bits

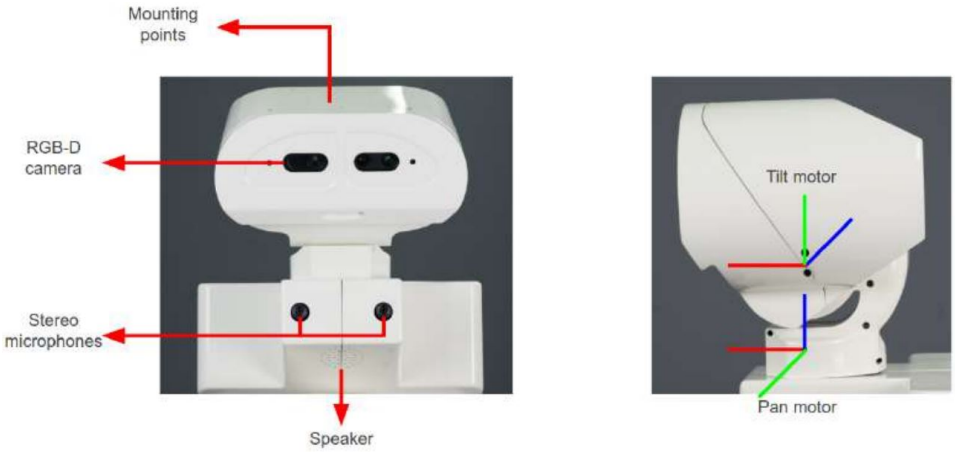
Tabulka 2.2: Vlastnosti naklápěcího mechanismu hlavy robota [L.20]

2.1.2 Ramena

TIAGo++ je vybavený dvěma rameny, které jsou sériovými manipulátory se 7 stupni volnosti. S rameny jsem pracoval více ve svém bakalářském projektu, kde jsem se snažil napodobit pózu člověka. Tyto ramena je možno ovládat různými způsoby a ty si představíme v dalších sekcích této práce.

2.1.3 Mobilní základna

Mobilní základna robota TIAGo++ je vybavena diferenciálním hnacím mechanismem a zároveň je v ní umístěn vestavěný počítač, baterie, napájecí konektory, laserový snímač vzdálenosti a WiFi přijímač pro zajištění bezdrátového připojení robota k řídícímu PC. Pomocí mobilní základny se robot může pohybovat po pracovním prostoru, a to až maximální rychlostí 1 $m \cdot s^{-1}$ dle tabulky 2.1. Mobilní základna je schopná rotace na místě bez nutnosti pohybu vpřed. Tohoto faktu využijeme při návrhu ovládání pohybu. V mobilní základně se nachází i výpočetní jednotka robota (IMU), která při jeho pohybu



Obrázek 2.2: Hlava robota s popisem jejích částí a umístěním souřadných systémů naklápěcího mechanismu [L.20]

Manufacturer	Orbbec
Model	Astra S
Field of view	60° H, 49.5° V, 73° D
Color stream modes	VGA 640x480 @ 30 fps
Depth sensor range	0.4 - 2 m

Tabulka 2.3: Specifikace kamery umístěné v hlavě robota TIAGo++ [L.20]

počítá podle naměřených rychlostí otáčení kol přibližnou polohu robota v pracovním prostoru vůči počátečnímu souřadnému systému *odom*.

2.2 Pracovní prostor

Pro správnou práci robota je nutné dodržet určité požadavky kladené na podobu a vlastnosti pracovního prostoru, ve kterém budeme s robotem pracovat. Hlavní požadavek je ten, že se v pracovním prostoru nevyskytují žádné překážky, do kterých by robot při svém pohybu mohl narazit nebo které by znemožnily detekovat člověka. Neboli robot musí mít vždy možnost člověka detekovat z místa, ve kterém se právě nachází bez dodatečného pohybu před jeho detekcí. Také je nutné dodržet minimální vzdálenost člověka od kamery tak, aby člověk byl v záběru kamery vidět celý. Tím se podstatně zvýší pravděpodobnost správné detekce člověka v obraze. Dle testování je optimání rozmezí vzdálenosti člověka od robota přibližně 2,5 - 4 metry.

Kapitola 3

Způsob řešení

3.1 Obecný postup

Pro dosažení cílů stanovených pro tuto práci je potřeba, aby byl robot schopen vykonávat řadu úkonů, které na sobě závisí. Jako první bychom si měli ověřit, že jsme schopni robota a jeho části ovládat pomocí teleoperace za pomocí ovladače či klávesnice. Následně je potřeba vyřešit způsob automatického ovládání mobilní základny robota, aby se mohl pohybovat po pracovním prostoru a přesunout se do určitého místa v něm. S tím zároveň souvisí schopnost otočení se na místě během fáze hledání člověka. Stejně tak důležitá je dovednost ovládat orientaci hlavy robota za účelem možnosti prohledávat své okolí a hledat člověka, od kterého následně bude přijímat příkaz k pohybu.

Když je robot schopen prohledávat svůj pracovní prostor, poté musí umět detekovat člověka v obraze pořízeném kamerou v hlavě. Člověk se ovšem nemusí nacházet přímo uprostřed záběru a tak musí robot sledovat člověka pohybem své hlavy a případně i celého zbytku těla tak, aby měl pokud možno člověka neustále uprostřed svého zorného pole. Detekce člověka v této fázi nemusí být přesná. Naopak je potřeba, aby byla rychlá a robot tak byl schopný člověka v zorném poli udržet i během pohybu sledované osoby.

Když robot v této fázi detekuje, že se člověk určitou dobu nepohnul, pro robota to znamená, že mu člověk zadává příkaz k pohybu a přesnějším způsobem detekuje pózu člověka a především to, kam ukazuje jeho ruka. Robot tedy musí být schopen z obrazu kamery určit, jak daleko se člověk

nachází a také místo, do kterého ukazuje. Pokud robot výpočtem dospěje k výsledku, že místo, do kterého člověk ukázal, je dosažitelné, pokusí se do tohoto místa dojet pomocí ovládání své mobilní základny. Jelikož chceme, aby celý proces byl opakovatelný, robot si spočítá, kterým směrem by se měl nacházet člověk poté, co robot dokončí svůj pohyb do zadaného místa. Robot se následně otočí do směru, ve kterém by se sledovaná osoba vyskytovala v případě, že se nepohnula ze svého místa. V tomto místě se vracíme zpět do fáze hledání člověka v pracovním prostoru, ovšem s mírnou výhodou znalosti předchozí jeho polohy.

Jak je vidět, výše popsaný sled úkonů se dá rozdělit do několika menších činností, které robot musí být schopen vykonávat. Na popis těchto dílčích úkolů se nyní v následujících sekcích zaměříme.

3.2 Ovládání hlavy robota

Jak již víme ze sekce 2.1.1, hlava robota je umístěná na naklápěcím mechanismu sestaveném ze dvou rotačních kloubů. Robot je tak schopen sledovat svůj pracovní prostor jednoduchým otáčením hlavy do stran. Hlavu budeme ovládat tak, že robotu pomocí číselné hodnoty řekneme, do jakého úhlu má svou hlavu natočit a robot do daného směru natočí hlavu. Stejným způsobem lze ovládat i naklánění hlavy nahoru a dolů v případě potřeby zaměření člověka do středu obrazu tak, aby na pořízeném snímku bylo vidět celé jeho tělo. Implementace výše popsaného způsobu ovládání je dále popsaná v sekci 4.6

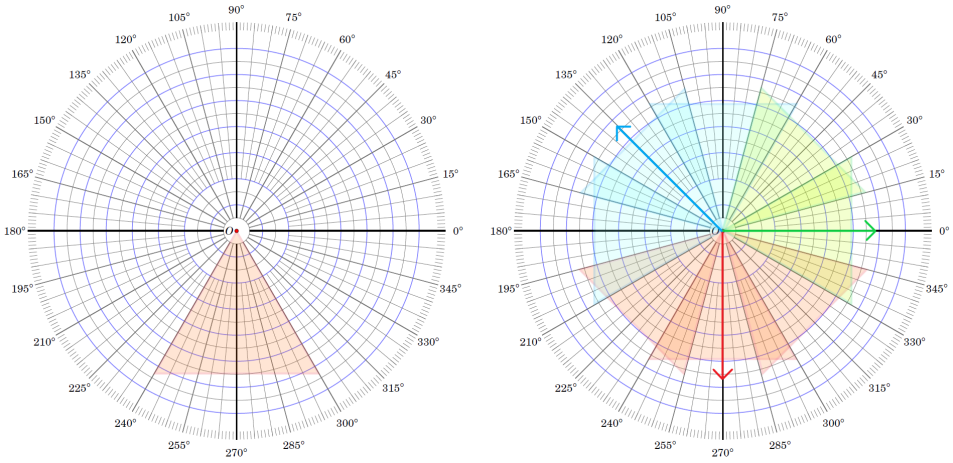
3.3 Rychlá detekce člověka v obraze

Když jsme schopni pohybovat hlavou robota, můžeme začít v pracovním prostoru hledat člověka. V této části procesu nepotřebujeme rozpoznávat příkazy člověka, protože ty budeme rozpoznávat až poté, co se člověk zastaví a nebude se dále pohybovat, čímž vydá robotovi pokyn k pohybu. Základní princip je takový, že robot pořídí snímek svého pracovního prostoru svou kamerou, ten zpracuje a pokud na něm objeví člověka, pokusí se pootočit hlavou tak, aby ho umístil do středu svého zorného pole. Pokud však na snímku člověka nedetekuje, pootočí svou hlavou na část pracovního prostoru, kterou předtím neměl ve svém zorném poli a opět analyzuje pořízený snímek. Když tímto způsobem neobjeví ve svém zorném poli člověka a prohledá tak

pracovní před sebou, robot se pak pomocí své mobilní základny otočí na místě (viz sekce 3.5) a zopakuje tak prohledání pracovního prostoru viditelného v novém směru. Pokud by ani po sekvenci prohledávání a otáčení na místě a pokrytí celého pracovního prostoru tímto způsobem nenašel člověka, robot se opět otočí do původního směru a tento cyklus se bude opakovat, dokud člověka neobjeví. Implementace této části řešení a způsob analýzy obrazu využívaný v této a následujících odstavcích je dále popsaný v sekci 4.7

3.3.1 Pokrytí pracovního prostoru zorným polem

Z tabulky 2.3 již víme, že zorné pole robota má šířku 60°. Toto zorné pole máme znázorněné na obrázku 3.1. Jelikož se robot bude nacházet uprostřed pracovního prostoru a postupně ho bude prohledávat, jednoduché řešení by mohlo být takové, že by se robot vždy pootočil svou hlavou o zmíněných 60° a k prohledání celého pracovního prostoru by tak stačilo 6 snímků. Takové řešení ovšem nebere v potaz možnost, že by se člověk vyskytoval v úhlech oproti původnímu směru robota přesně v násobcích 60°. V takovém případě by mohlo nastat, že by robot viděl polovinu člověka na jednom záběru a druhou polovinu na snímku druhém a rozpoznání člověka by pak v takovém případě nemuselo zafungovat správně. Tuto možnou chybu můžeme eliminovat tak, že robot bude otáčet hlavou po menších úsecích a okraje zorného pole jedno snímku pak prohledá na i na snímcích následujících. Případné uříznutí a špatná detekce člověka při pokrytí s přesahy tak nemůže nastat. Pro náš případ jsme zvolili otáčení po 45° a k pokrytí celého prostoru tak budeme potřebovat 8 snímků.



Obrázek 3.1: Vlevo - zorné pole robota při pořízení snímku, Vpravo - Pokrytí pracovního prostoru při hledání člověka. Červená šipka - výchozí natočení, modrá šipka - směr robota po prvním otočení základny, zelená šipka - směr robota po druhém otočení základny.

3.3.2 Vyhledávací algoritmus

Čím rychleji člověka v pracovním prostoru najdeme, tím lépe. Tím pádem chceme minimalizovat dobu, kterou bude robotu trvat otočení základny či hlavy před pořízením snímku. Tomuto požadavku bude také přizpůsobena sekvence pohybů, která zajistí vyhledání člověka. K jejímu následujícímu popisu budu používat a odkazovat se na pravý graf na obrázku 3.1. Na něm jsou zakresleny barevné šipky znázorňující směr, do kterého se otáčí základna robota. Následně jsou na něm zobrazená zorná pole podle těch barev, ve kterém natočení základny je robot pořídil.

Pokud je robot schopný pokrýt novou část pracovního prostoru pouhým otočením hlavy, otočí ji do nového směru, pořídí snímek a ten následně analyzuje. Pokud již pohyby hlavy při jednom natočení základny pokryl všechny možné úhly, natočí se základnou do jiného, předem určeného směru a v něm opět provede sérii snímků. Tyto dva kroky se boudou opakovat dokud se neprohledá celý pracovní prostor kolem robota a případně se celý proces bude opakovat, dokud člověka na snímcích nenajdeme. Tuto nastavenou sekvenci pohybů k prohledávání budeme vykonávat vždy, když člověk zmizí ze záběru robota. Robot jí tak musí být schopen vykonat bez ohledu na původní orientaci své základny před začátkem této sekvence. Sekvence pohybů vedoucích k prohledání pracovního prostoru je stále stejná a sled pohybů se tak nebude měnit. Během tohoto sledu příkazů platí, že mezi jednotlivými snímky pohne robot vždy buď jen hlavou, či základnou, nikdy oběma částmi najednou. Tím se minimalizuje čas strávený pohybem robota mezi pořizováním snímků. Pseudokód pro tento proces je předepsaný v algoritmu 1

3.3.3 Zaměření člověka do středu obrazu

Po nalezení člověka sekvencí pohybů popsanou v algoritmu 1 chceme člověka udržet přibližně uprostřed zorného pole kamery robota. Dokud tomu tak nebude, uložíme si aktuální obraz z kamery robota, ten analyzujeme a následně budeme spouštět funkci popsanou v algoritmu 2. Ta zařídí, že pokud by kvůli pohybu člověka po pracovním prostoru měl robot pro zacentrování na střed těla otočit hlavou o úhel od přímého směru větší než 70° (limit kloubu robota), robot se místo toto otočí celou základnou do požadovaného směru a hlavou bude opět směřovat přímo před sebe (tedy bude otočena v úhlu 0°)

Algoritmus 1: `find_human()`

```
step ← 0;
tries ← 0;
θ ← mobile_base.rotation;
while not human_info.is_found() do
    if step = 0 &
     |latest_joint_states.return_head_joint_potitons()[0]| >
     0.01 then
        head_position ←
         latest_joint_states.return_head_joint_potitons();
        base_rotation ← base_rotation + head_positon[0];
        move_head(0°, 0°, wait_for_result = False);
        move_base(Point(), degrees(θ), on_place = True);
    else if step = 1 then
        move_head(45°, 0°);
    else if step = 2 then
        move_head(−45°, 0°);
    else if step = 3 then
        move_base(Point(), degrees(θ) − 135°, on_place = True);
    else if step = 4 then
        move_head(0°, 0°);
    else if step = 5 then
        move_head(−45°, 0°);
    else if step = 6 then
        move_base(Point(), degrees(θ) + 90°, on_place = True);
    else if step = 7 then
        move_head(0°, 0°);
    else if step = 8 then
        move_base(Point(), degrees(θ), 0°, on_place = True);
        step ← 0;
    end
    tries ← 0;
    while not human_info.is_found() & tries < 3 do
        analyze_picture_with_mediapipe(step);
        tries ← tries + 1;
    end
    step ← step + 1;
end
```

Algoritmus 2: `center_camera_on_human()`

$\theta_1, \theta_2 \leftarrow latest\_joint\_states$.`return_head_joint_positions`();
$\alpha_{hor} \leftarrow \theta_1 + human\_info.final\_robot\_rotation.x$;
$\alpha_{ver} \leftarrow \theta_2 + human\_info.final\_robot\_rotation.y$;
if $|\alpha_{hor}| < 70°$ **then**
  `move_head`$(\alpha_{hor}, \alpha_{ver})$;
else
  `move_head`$(0°, \alpha_{ver}, wait\_for\_result =$ **False**$)$;
  `move_base`(`Point`$(), mobile\_base.rotation + \alpha_{hor}, on\_place =$ **True**$)$;
end

3.4 Přesná detekce člověka v obraze

Pokud chceme z obrazu kamery určit, kam člověk ukazuje, budeme muset obraz z kamery zpracovávat jinak než doposud, a to s použitím dvou předtrénovaných neuronových sítí (viz sekce 4.9). Z obrazu kamery, který pořídíme, nejprve vytvoříme 2D kostru člověka pomocí první neuronové sítě a poznamenáme si její souřadnice. Stejný obraz pak pošleme společně se souřadnicemi z první sítě do druhé, ta nám z nich vytvoří 3D kostru člověka a souřadnice klíčových bodů celého těla. Ty následně budeme zpracovávat tak, abychom z nich dostali místo, na které člověk ukazuje pravou rukou v souřadném systému *odom*.

3.4.1 Souřadné systémy

Pro to, abychom byli schopni člověka nějakým způsobem umístit do souřadného systému robota, si musíme zadefinovat souřadné systémy, jimiž budeme popisovat pozici člověka, robota i finálního místa dojezdu v pracovním prostoru. Jelikož řešíme pohyb po pracovní ploše, uvažujeme zjednodušený systém souřadnic ve 2D. Souřadnice základny robota tak budeme uvažovat bez její výšky a body dojezdu budou také převedeny do roviny pracovního prostoru. Implementace výpočtu transformace souřadných systémů je popsána v sekci 3.4.3.

Souřadný systém *odom*

Souřadný systém *odom* je souřadný systém, který má svůj počátek v místě, kde se robot při svém zapnutí a inicializaci nacházel. V tomto souřadném systému jsou pak ROSem poskytována polohová data základny robota, která následně používáme k zadání cílového místa pohybu pro robota. Jelikož výsledkem této práce má být to, že je robot chopen dojet na určené místo, musíme každý nalezený bod dojezdu transformovat do tohoto souřadného systému. Tím pádem je tento souřadný systém pro nás shodný se světovým souřadným systémem a budeme ho tak i interpretovat.

Souřadný systém *base_footprint*

Tento souřadný systém je spojen se středem základny robota a tím pádem není statický. Budeme ho používat jako mezikrok během transformace finálního bodu dojezdu a pozice člověka do souřadného systému *odom*.

Souřadný systém spojený s člověkem *human*

V tomto souřadném systému jsou uvedené souřadnice všech nalezených důležitých bodů člověka (viz sekce 4.9.2). Také je v něm prováděn výpočet vektoru směru ukazování pravé ruky a místa dojezdu robota.

3.4.2 Určení místa dojezdu

Jelikož nalezené body budou uvedené v souřadnicovém systému *human*, známe všechny tři jejich prostorové souřadnice a můžeme mezi dvěma body na ruce (ramenem a zápěstím) nalézt směrový vektor. Ten využijeme ke zjištění parametrického popisu přímky mezi těmito body a k výpočtu průsečíku této přímky se zemí následujícím postupem:

1. Uvažujeme dva body A a B, mezi kterými chceme najít přímku jimi procházející. Souřadnice bodů si můžeme označit následovně

$$\mathrm{A} = [A_x,\ A_y,\ A_z] \qquad \mathrm{B} = [B_x,\ B_y,\ B_z]$$

2. Mezi body nalezneme směrový vektor $\vec{v}$ z bodu A do bodu B jako

$$\vec{v} = \mathrm{B} - \mathrm{A} = (B_x - A_x,\ B_y - A_y,\ B_z - A_z) = (v_x,\ v_y,\ v_z)$$

3. Když známe směrový vektor, můžeme pomocí něj a bodu A sestavit parametrický popis hledané přímky p v prostoru jako

$$p : [A_x,\ A_y,\ A_z] + t \cdot (v_x,\ v_y,\ v_z)$$

4. Nyní potřebujeme nalézt bod C takový, že leží na přímce p a zároveň je jeho z-ová souřadnice rovna nule (nachází se na zemi). Pro zjištění tohoto bodu nejprve musíme nalézt parametr t z rovnice:

$$C_z = A_z + t \cdot v_z$$

ze které si pro $C_z = 0$ vyjádříme parametr t jako

$$t = -\frac{A_z}{v_z}$$

5. Zbylé souřadnice si vypočítáme dle stejného předpisu jako pro souřadnici v ose z se znalostí parametru t. Výsledný bod C má tedy následující prostorové souřadnice:

$$\mathrm{C} = \left[A_x - \frac{A_z}{v_z} \cdot v_x,\ A_y - \frac{A_z}{v_z} \cdot v_y,\ 0\right]$$

Takto nalezený průsečík se zemí Poté musíme následně transformovat ze souřadného systému *human* do *odom*.

3.4.3 Transformace mezi souřadnými systémy

K transformaci bodu z jednoho souřadného systému do druhého použijeme rotační a translační matice. Pokud máme polohový vektor $\vec{r_1}$ a chceme ho transformovat do souřadného systému 0, použijeme k tomu rotační matici $\mathbf{R_1^0}$ a translační matici $\mathbf{T_1^0}$ jako

$$\vec{r_0} = \mathbf{T_1^0} \cdot \mathbf{R_1^0} \cdot \vec{r_1}$$

Transformace budeme provádět v následujícím sledu souřadných systémů

$$\textit{human} \quad \rightarrow \quad \textit{base\_footprint} \quad \rightarrow \quad \textit{odom}$$

Jelikož budeme provádět transformace bodů již v rovině pracovní plochy, budeme využívat translační a rotační matice v rovině a souřadnice bodů budeme rozšiřovat pro správný výpočet násobením matic.

Obecný tvar rotační matice v rovině představující rotaci o úhel α je

$$\mathbf{R}(\alpha) = \begin{pmatrix} \cos(\alpha) & -\sin(\alpha) & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Obecný tvar translační matice reprezentující posunutí v osách x a y má tvar

$$\mathbf{T}(x,\ y) = \begin{pmatrix} 1 & 0 & x \\ 0 & 1 & y \\ 0 & 0 & 1 \end{pmatrix}$$

3.4.4 Nalezení vzdálenosti člověka od robota

Ke správnému provedení transformace mezi systémy *human* a *base_footprint* je potřeba znát vzdálenost člověka od základny robota. K tomu využijeme hloubkového obrazu z kamery robota. Robot disponuje RGB-D kamerou, která je schopna s pomocí dalších senzorů pořídit hloubkový obraz, kde v každém pixelu je zakódovaná vzdálenost daného místa v metrech. Pokud předtím analyzujeme RGB obraz a nalezneme v něm člověka, poté známe i polohu středu těla člověka v obraze a podíváme se na stejné místo v hloubkovém obraze. Tím se dozvíme vzdálenost člověka od kamery. Jelikož se však kamera nenachází přímo v ose z souřadného systému *base_footprint*, tento posun od středu přidáme ke vzdálenosti zjištěné z obrazu a získáme tak vzdálenost člověka od základny robota.

3.5 Pohyb robota po pracovním prostoru

Robot se po pracovním prostoru může pohybovat pomocí své mobilní základny (viz sekce 2.1.3). Při pohybu si pomocí své IMU jednotky určuje odometrii. Díky té víme, kde se robot v pracovním prostoru nachází. Odometrie je vztažena k místu, kde byl robot inicializován a tento souřadný systém se jmenuje *odom* (viz sekce 3.4.1). V něm budeme veškeré pohyby robota vykonávat. Pohyb robota se bude ovládat pomocí příkazů rychlostí pohybu v jednotlivých osách robotické základny. K tomu, abychom byli schopni robota nasměrovat do určitého místa dle požadovaných souřadnic v souřadném systému *odom*, budeme potřebovat jednoduchý regulátor pohybu. Tomu předáme požadované souřadnice a on se postará o pohyb robota. Pseudokód předepsaný pro tento regulátor je níže popsaný algoritmu 3 a jeho implementace je popsaná v sekci 4.8

Algoritmus 3: move_base(*goal*, θ, *on_place*)

```
r ← rospy.Rate(20000);
speed ← Twist();
while not on_place do
    x ← goal.x − mobile_base.position_x;
    y ← goal.y − mobile_base.position_y;
    if √(x² + y²) < 0.1 then
        speed.linear.x ← 0;
        speed.angular.z ← 0;
        mobile_base.pub_velocily_command.publish(speed);
        break;
    end
    φ ← math.atan2(y, x);
    if φ − mobile_base.rotation > 0.1 then
        speed.linear.x ← 0;
        speed.angular.z ← 0.5;
    else if φ − mobile_base.rotation < −0.1 then
        speed.linear.x ← 0;
        speed.angular.z ← −0.5;
    else
        speed.linear.x ← 0.2;
        speed.angular.z ← 0;
    end
    mobile_base.pub_velocily_command.publish(speed);
    r.sleep();
end
while |θ − mobile_base.rotation| > 0.01 do
    if θ − mobile_base.rotation > 0.01 then
        speed.angular.z ← 0.5;
    else if θ − mobile_base.rotation < −0.01 then
        speed.angular.z ← −0.5;
    else
        speed.angular.z ← 0;
        break;
    end
    r.sleep();
end
```

Kapitola 4

Implementace řešení

4.1 Přehled sekce

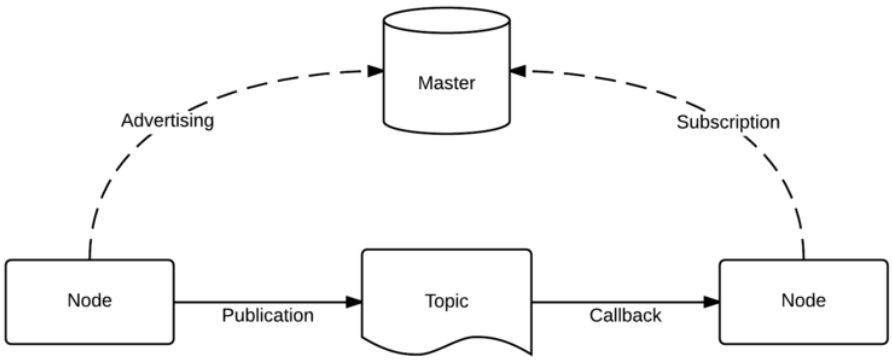
Tato sekce se zaměřuje na samotné provedení teoretického návrhu řešení v kapitole 3, popisuje použitý systém pro ovládání a komunikaci s robotem ROS [OR21], simulační prostředí Gazeboo [OR] využité v prvotních fázích návrhu ovládání jednotlivých částí robota. Dále též zmiňuje způsob komunikace mezi běžícími procesy, které nepoužívají ROS, použité neuronové sítě AlphaPose [FLT+22] a MotionBERT [ZML+22] spolu s balíkem knihoven pro rychlou analýzu obrazu při vyhledávání člověka MediaPipe [LTN+19]. Také obsahuje popis jednotlivých problémů, které bylo při návrhu řešení překonat a jakým způsobem byly problémy vyřešeny. Všechny zmiňované skripty jsou dostupné na GitHubu pod odkazem [Pro23]. Jednotlivé části vypracování budou popisovány v pořadí, ve kterém jsou během běhu programu využívány.

4.2 ROS

Robotic Operating System (ROS) je open-source balík softwarových knihoven uzpůsobený práci v oblasti robotiky, umožňující snadnou komunikaci mezi jednotlivými částmi robotických systémů. Pomáhá vývojářům s návrhem robotických aplikací a obsahuje řadu nástrojů pro opravy chyb, vizualizaci, správu systému a další. Podrobná dokumentace k tomuto softwaru je k

dispozici na webových stránkách [ros]. Tato práce je založená především na práci s ROSem a pomocí něj byla zprostředkována komunikace mezi programem a robotem jak v simulačním prostředí, tak i při práci se skutečným robotem.

Pomocí ROSu je možné komunikovat mezi více programy ve stejné aplikaci a tyto programy pak v kontextu ROSu nazýváme *nodes*, neboli uzly. K vytvoření komunikačního kanálu mezi dvěma programy (uzly) je potřeba hlavního uzlu, který se nazývá ROS Master. Po zapnutí se programy registrují u hlavního uzlu a komunikace mezi ostatními uzly je realizována pomocí asynchronních kanálů, kterým se říká *topics* (tedy témata). Pomocí nich je možné z robota dostávat informace o jeho kloubech, rychlosti, poloze v pracovním prostoru nebo také vyčítat obraz z jeho kamer. Zároveň je mu možné odesílat příkazy se souřadnicemi natočení jeho kloubů či požadované rychlosti pohybu jeho základny. Právě proto ve své práci budu tento způsob komunikace často využívat k přenosu informací do programů a odesílání příkazů do robota (schéma na obrázku 4.1). Abych z daného topicu byl schopný dostávat data, musím si definovat odběratele `rospy.Subscriber` na daný topic, typ zprávy, kterou chceme přijímat, a funkci, kterou zavoláme vždy při přijetí nové zprávy. Pokud chceme do topicu posílat data, musíme si vytrořit objekt typu `rospy.Publisher` opět definovaný názvem topicu a typem zprávy k odesílaní. Tu pak odešleme pomocí metody `publish`



Obrázek 4.1: Komunikační schéma mezi jednotlivými uzly v ROSu [Noe16]

4.2.1 Další způsoby komunikace mezi programy

Jelikož ke zprovoznění naší práce je zapotřebí mít spuštěné 4 Python skripty, mezi kterými bude probíhat komunikace, bylo zapotřebí vymyslet způsob sdílení dat mezi běžícími procesy za účelem jejich zpracování v daném skriptu. Pro vyřešení tohoto problému mi byly mým vedoucím práce poskyt-

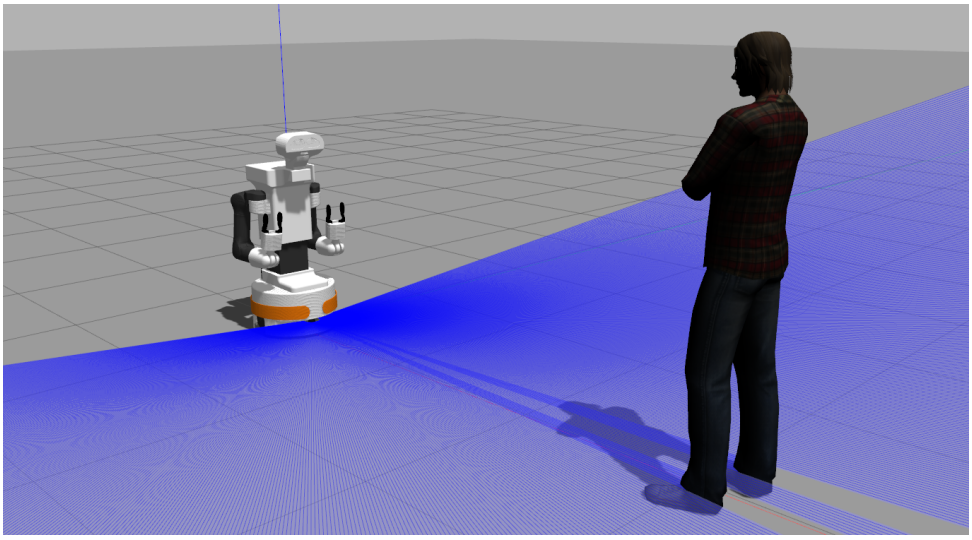
nuty skripty `service_client.py` a `service_server.py` využívající knihovny `zmq` a `cloudpickle`, ve kterých jsou definovány třídy `ServiceClient` a `ServiceServer`. Ty fungují jako klient a server, co reaguje na požadavek od klienta. Klient odešle požadavek serveru s daty ve struktuře slovníku, server je zpracuje dle potřeby a navrací klientovi opět slovník s případnými výsledky. Komunikace je prováděna protokolem TCP a probíhá přes LAN. Tím je umožněna komunikace i mezi skripty využívající Python 2 (hlavní skript `main.py` ovládající robota) a těmi využívající Python 3 (skripty `mediapipe_image_server.py`, `alphapose_image_server.py` a `motionbert_image_server.py` zprostředkovávající analýzu obrazu z kamery robota. Jelikož lze tímto způsobem přenášet jen určité typy objektů, obrazová data z kamery musím před jejich odesláním zakódovat do objektu typu `bytes` pomocí knihovny `base64`, který lze tímto způsobem přenášet. Na straně serveru obrazová data opět rozkódujeme a dále budeme zpracovávat.

4.3 Simulační prostředí Gazeboo

Simulační prostředí Gazeboo [OR] slouží k vizualizaci a simulaci ovládání v robotických aplikacích. Každá část programu byla otestována v tomto simulačním prostředí. Do prostředí je možné vkládat různé objekty, například i model člověka, který jsem používal pro otestování funkčnosti algoritmu pro nalezení člověka v pracovním prostoru. Gazeboo se nainstaluje spolu s ROSem a prostředím pro TIAGo++ pokud budeme postupovat podle návodu na webových stránkách [Rob23] a během mé práce na této bakalářské práci jsem toto simulační prostředí často používal.

4.4 Teleoperace a připojení k fyzickému robotu

Jedním z bodů vypracování této práce je ověření teleoperativního ovládání robota pomocí ovladače či klávesnice. Robota je možné ovládat pomocí Web Commanderu ve webovém prohlížeči. V tomto rozhraní jsou zároveň vidět všechny právě dostupné a běžící procesy. Například po zapnutí fyzického robota se automaticky zapne `head_manager`, jehož práce je otáčet automaticky hlavou do směru, kterým se robot pohybuje. Pro naši práci potřebujeme ovládat natočení hlavy dle rozpoznaného obrazu, tudíž je nutné tento automatický systém při startu robota deaktivovat, jinak bychom nebyli schopni hlavu ovládat. Ve webovém rozhraní je možné ovládat každý kloub robota jednotlivě, dát povel robotu k zaujmutí přednastavených póz a také ovládat



Obrázek 4.2: Robot TIAGo++ v simulačním prostředím společně s modelem člověka

pohyb základny robota.

Pokud bychom chtěli teleoperativně ovládat robota i jinak než přes webové rozhraní, k dispozici je také ovladač, jehož levou páčkou je možné ovládat pohyb základny dopředu a dozadu, pravou páčkou pak zase otáčení doleva a doprava. K dispozici je také ovládání natočení hlavy pomocí tlačítek na pravé části ovladače. Ovládací schéma je vidět na obrázku 4.3

Obrázek 4.3: Ovládací schéma ovladače pro teleoperaci robota.

4.4.1 Postup připojení k robotu

Robota je možné spustit červeným tlačítkem na levé zadní části základny robota a následně stiskem zeleného tlačítka napravo od něj. Abychom byli schopni robota ovládat pomocí počítače a odesílat mu příkazy bezdrátově, musí být v ovládacím počítači definován název robota společně s jeho IP adresou v souboru `etc/hosts`. Název konkrétního robota, na kterém jsem bakalářskou práci testoval byl `tiago-114c`, tudíž do zmiňovaného souboru je nutné připsat řádek `"192.168.130.114 tiago-114c"`, kde zmíněná IP adresa je adresa robota. Tento alias je pak možné využívat pro připojení k Web Commanderu či přímo k robotu přes `ssh`. Web Commander je pak dostupný na adrese `http://tiago-114c:8080`. Dalším krokem je definice proměnných *ROS_MASTER_URI* a *ROS_IP* v každém terminálu, který bude přes ROS libovolným způsobem komunikovat s robotem. Zadávané hodnoty během této práce:

```
export ROS_MASTER_URI=http://192.168.130.114:11311
export ROS_IP=<moje_ip_adresa>
```

Správné fungování komunikace mezi robotem a řídícím počítačem je možné ověřit například pomocí příkazu `rostopic list` vypisující všechny dostupné topics a následným nahlédnutím například do obsahu zprávy s transformacemi pomocí příkazu `rostopic echo /tf`.

4.5 Popis vytvořených objektů

Pro usnadnění práce s proměnnými a zpřehlednění kódu jsem si v průběhu práce vytvořil několik tříd v souborech `main.py` a `classes.py`, které zde stručně popíšu.

4.5.1 Objekty definované v souboru `classes.py`

`LocationPoint`

Tato třída má funkci reprezentace dvourozměrných souřadnic v různých souřadných systémech, např. souřadnice obrazových bodů a souřadnice polohy

robota.

`Location3DJoint`

Tento objekt slouží pro reprezentaci trojrozměrných souřadnic v souřadném systému *human*, které do tohoto objektu ukládáme v metodě `HumanInfo.create_joints_dict`

`HumanInfo`

Třída slouží pro ukládání informací o nalezeném člověku v obraze. Ukládá si informace o tom, zda byl člověk v záběru nalezen, zda se nachází uprostřed záběru kamery, kde se nachází střed jeho těla v relativních obrazových souřadnicích, potřebné rotace kloubů hlavy robota pro zacentrování na člověka, jeho vzdálenost od robota a i výsledné významné body na těle člověka z AlphaPose i MotionBERT. Ve své metodě `create_joints_dict` si výsledné souřadnice bodů z MotionBERTa přeskládává do slovníku pro jednodušší práci s nimi. Metoda `change_reference_of_joints` zase posouvá počátek souřadného systému *human* z původní polohy přibližně uprostřed těla do místa na podlaze pod středem těla člověka, který je určen z nalezených klíčových bodů pravé a levé pánve.

4.5.2 Objekty definované v souboru `main.py`

`Camera`

Tento objekt si pomocí topicu `/xtion/rgb/camera_info` načte do programu parametry kamery v robotu, včetně výšky a šířky obrazu. Metoda tohoto objektu `find_distance_to_human` si za pomocí nalezených souřadnic středu těla člověka a hloubkového obrazu z kamery, který je dostupný na topicu `xtion/depth_registered/image_raw`, spočítá vzdálenost člověka od základny robota. Hloubkový obraz si ze zprávy pomocí objektu `CvBridge` převedeme na pole, jehož prvky jsou vzdálenosti jednotlivých pixelů obrazu od kamery v metrech. K této vzdálenosti přičteme vzdálenost souřadného systému kamery `xtion_rgb_optical_frame` od základny robota `base_footprint` a tím získáme vzdálenost člověka od základny robota.

ImageConverter

Tento objekt slouží pro práci s obrazem z kamery, který získáme odebíráním zpráv z topicu `/xtion/rgb/image_raw` a konverzí metodou `CvBrdge.imgmsg_to_cv2`. Obraz z kamery zároveň zakódujeme pomocí modulu `base64` do objektu typu `bytes`, abychom obraz mohli odeslat do ostatních skriptů. Také obsahuje metodu `get_current_view`, která slouží k zobrazování obrazových dat v okně a jejich ukládání na disk.

LatestJointStates

Tento objekt byl převzatý dokumentace ROSu na webové stránce [Thi10] a upraven pro potřeby mé práce. Objekt odebírá zprávy z topicu `joint_states`, na kterém jsou dostupné informace o poloze, rychlosti a úsilí všech kloubů robota. Jelikož jsou pro nás důležité především pozice kloubů hlavy robota, připsaná metoda `return_head_joints_positions` nám vrátí právě ty.

BasePositionWithOrientation

Tato třída slouží pro uchovávání odometrických dat pro reprezentaci polohy základny robota v souřadném systému *odom* jako jsou její x-ová a y-ová souřadnice, natočení, lineární a úhlová rychlost. Zároveň obsahuje publishera na topic `/mobile_base_controller/cmd_vel`, kterým budeme posílat zprávy s požadovanými rychlostmi základny robota. Kód který ukládá odometrická data, byl inspirovaný kódem na webu [Hua17], který byl také základem pro ovládání základny robota ve funkci `move_base`

4.6 Implementace ovládání hlavy robota

Ovládání hlavy robota je implementováno v souboru `main.py` ve funkci `move_head`. Pro hlavu robota vytvoříme v ROSu objekt typu `SimpleActionClient`, pomocí kterého můžeme do vybraného topicu odesílat příkazy k vykonání pohybu hlavy do námi požadovaných úhlů. Topic pro ovládání hlavy se jmenuje `/head_controller/follow_joint_trajectory`

4.6.1 Popis funkce `move_head`

Vstupními parametry do funkce jsou samotný akční klient pro hlavu robota, požadované úhly natočení hlavy zadávané ve stupních a volitelný parametr *wait_for_result*. Jako první si ve funkci vytvoříme objekt `JointTrajectory`, který bude sloužit k uchování informace o požadované trajektorii pohybu hlavy robota. My chceme každým zavoláním této funkce pohnout hlavou do určitého místa, tedy naše trajektorie se bude skládat pouze z jednoho finálního bodu. Trajektorii je také nutné přidělit informaci o čase, aby klient věděl, kdy má danou trajektorii začít vykonávat, názvy úhlů, které v trajektorii chceme ovládat, a bod, do kterého chceme hlavu otočit, definujeme jako objekt typu `JointTrajectoryPoint`. Bodu je nutné definovat jeho pozice (souřadnice kloubů, do kterých se mají klouby otočit), rychlosti průjezdu těmito body a čas, do kterého se má pohyb dokončit.

Následně si definujeme cíl jako objekt typu `FollowJointTrajectoryGoal`, kterému přiřadíme vytvořenou trajektorii a časovou toleranci, do kterého se má hlava robota dostat do požadovaného bodu. Cíl odešleme do robota pomocí metody `send_goal` akčního klienta. Pokud by byla nastavená proměnná *wait_for_result* na hodnotu **True**, program bude čekat do konce vykonávání pohybu a následně zhodnotí, zda se pohyb hlavy úspěšně dokončil či nikoliv.

4.7 Rychlá detekce člověka v obraze

Když jsme schopni ovládat hlavu robota, můžeme s ní začít prohledávat pracovní prostor. K tomu bude potřeba vyčítat obraz z kamery umístěné v hlavě robota, zpracovat obraz pomocí knihovny MediaPipe [LTN+19] a případně nalezeného člověka udržet v zorném poli. RGB obraz z kamery můžeme dostat z topicu `/xtion/rgb/image_raw` jako zprávu typu `Image` a následně jí pomocí objektu `CvBridge` převést na obrazová data ve formátu knihovny OpenCV. Postup vyhledávání člověka byl již popsán v algoritmu 1 a způsob udržení člověka v zorném poli zase v algoritmu 2. Zbývá nám tedy popsat jen způsob zpracování obrazu pomocí MediaPipe.

4.7.1 MediaPipe

MediaPipe [LTN+19] je knihovna s nástroji pro adaptaci umělé inteligence a strojového učení pro různé programátorské úkoly. Jedním z připravených nástrojů je detekce pózy člověka v obraze. Toto řešení budeme používat při prohledávání pracovního prostoru i kontrole centrování na člověka. Zpracování obrazu pomocí této knihovny je implementováno v souboru `mediapipe_image_server.py` a funkci `find_human_in_image_by_mediapipe`, kterou si teď popíšeme. Vizualizaci nalezených bodů pomocí této knihovny je na obrázku 4.5. Ve své práci využívám starší verzi `0.8.2`

Popis funkce `find_human_in_image_by_mediapipe`

Vstupem do funkce je objekt *human_info* typu `HumanInfo` a obraz přijatý od klienta *cv2_img*. Před každým rozpoznáním si v objektu *human_info* vymažu data o předchozím snímku. Poté si načtu řešení *pose* pro zpracovávání obrázků a ten zpracuji příkazem *pose*.`process()`. Pokud při zpracování nebyly nalezeny žádné orientační body na těle člověka (tzn. člověk na obraze nebyl nalezen), vrátím klientovi prázdný objekt *human_info*. Ovšem při nalezení člověka najdu střed jeho těla pomocí funkce `get_human_center_location` a potřebnou rotaci hlavy robota pro zacentrování kamery na člověka pomocí funkce `get_needed_robot_rotation`. MediaPipe při nalezení člověka vrací souřadnice významných bodů na lidském těle v pixelových souřadnicích. Střed těla je bod těžiště z nalezených poloh ramen a kolen.

4.8 Ovládání pohybu robota v pracovním prostředí

Když člověka nenajdeme pouze pomocí otáčení hlavy, budeme muset otočit celým robotem pomocí jeho základny. Pohyb základny je ovládán opakovaným odesíláním zprávy obsahující data o požadované rychlosti v daném směru, která je objektem typu `Twist` a odesíláme jí robotovi na topic `/mobile_base_controller/cmd_vel`. Zprávu s požadovanými rychlostmi je nutné posílat opakovaně, aby robot věděl, že nedošlo k výpadku komunikace během vykonávání příkazu. Robot by pak z důvodu bezpečnosti sám zastavil. Přesun robota do určitého místa v pracovním prostoru a otáčení robota na místě do požadovaného směru je realizováno v souboru `main.py` a funkci `move_base`, tu teď stručně popíšu.

4.8.1 Popis funkce `move_base`

Vstupem do funkce je objekt *goal* typu `Point`, kterým definujeme souřadnice cílového místa pohybu základny, proměnná *final_angle_in_degs*, která nese informaci o finálním natočení základny robota, a parametr *on_place*, který nám říká, zda se má základna pohybovat do místa *goal*, či se jen otočit na místě do finálního úhlu. Ten se na začátku funkce případně přesune do rozmezí (-180°; 180°). Všechny parametry jsou vztažené k souřadnému systému *odom*.

Pokud se má robot přesunout do určitého místa (parametr *on_place* = **False**), v každém kroku se spočítají vzdálenosti základny robota od finálního místa v osách x a y, spočítáme si úhel mezi těmito dvěma body a pokud zjistíme, že robot není natočený čelem k finálnímu místu, tak se základna otáčí na místě tím směrem, kterým je to rychlejší a otočí se do směru cíle. Poté už základna jede přímo rovně k cíli. Do robota se odesílá informace o požadované lineární a úhlové rychlosti v metrech za sekundu. Pro plynulý pohyb základny robota je nutné zprávu do výše zmiňovaného topicu odesílat velice často. experimentálně bylo zjištěno, že odesílání zpráv rychlostí 10 - 20 tisíc za vteřinu je dostatečné k plynulému pohybu. Pokud robot neobdržuje pokyny k pohybu takto často, pohyb robota je velice sekaný, absenci rychlého sledu příkazů robot interpretuje z bezpečnostních důvodů jako příkaz k zastavení.

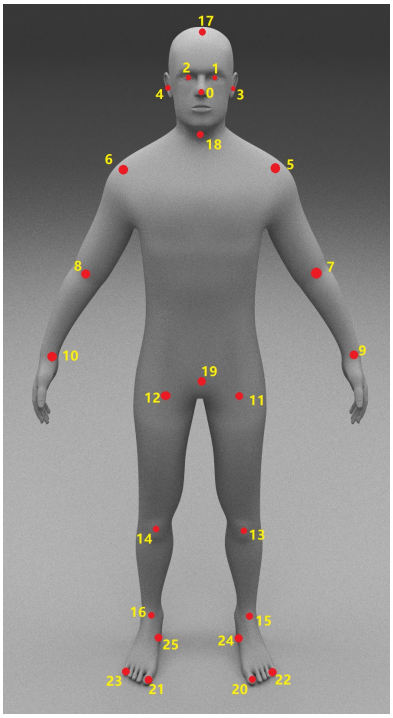
Když se robot dostane do finálního místa nebo byl nastavený parametr `on_place` na **True**, robot se na aktuálním místě začne otáčet do finálního směru, dokud není do finálního směru otočen. Mezi fází přesunu na místo a otáčení na místě se čeká, než se robot zastaví na místě, a až poté se pokračuje v programu. Pseudokód je ukázán v algoritmu 3

4.9 Přesná detekce člověka v obraze

Pokud se člověk nachází uprostřed záběru kamery robota, poté můžeme na povel člověka začít přesně analyzovat polohu významných bodů na jeho těle. K tomu využijeme předtrénované neuronové sítě. Obraz z kamery robota si nejprve předzpracujeme pomocí AlphaPose. Tato síť nám vrátí slovník s informacemi o 2D kostře člověka s klíčovými body určenými datasetem Halpe (viz obrázek 4.4). 2D kostru člověka z tohoto modelu potřebujeme pro druhou použitou neuronovou síť jménem MotionBERT. Ta pomocí výsledků z AlphaPose dokáže 2D kostru člověka převést na 3D kostru. Trojrozměrné souřadnice jsme pak schopni dále zpracovávat.

4.9.1 AlphaPose

Neuronová síť AlphaPose [FLT+22] je přesný estimátor pózy člověka v obraze. Výsledná poloha člověka je zakódována ve 26 klíčových bodech (viz obrázek 4.4). AlphaPose vyžaduje ke svému fungování virtuální prostředí Anaconda a Python 3.7. Pro tuto práci jsem stáhl zdrojový kód k AlphaPose [MS23] a v něm jsem si upravil několik souborů tak, aby bylo možné provádět rozpoznávání opakovaně a reagovat na požadavek o rozpoznání od klienta. Hlavní skript, který spouštím serverovou část rozpoznávání je `alphapose_image_server.py`.



Obrázek 4.4: Model člověka z datasetu Halpe vytvořený pro síť AlphaPose s polohami výsledných bodů [FLT+22]

Popis souboru `alphapose_image_server.py`

Na začátku tohoto skriptu jsou nastavené určité parametry pro využití AlphaPose pro jeden snímek a upravené tak, aby bylo možné zpracovávat informace přijaté pomocí `ServiceServer` objektu ve skriptu `service_server.py` (viz sekce 4.2.1) Parametry pro nastavení si uložím, načtu si konfiguraci pro použitý předtrénovaný model. Poté si inicializuji objekt *pose* typu `SingleImageAlphaPose` vytvořený ve zdrojových kódech k AlphaPose a také samotný server, který bude čekat na příkazy od klienta. Při obdržení poža-

davku na zpracování obrazových dat se spustí funkce `process_image`. Ta vytvoří případnou složku s výsledky analýzy, dekóduje si obdržená obrazová data a ta zpracuje pomocí metody *pose*.`process()`. Pomocí upravené metody *demo*.`writeJson` pak dostaneme slovník s výsledky analýzy a ten odešleme zpět klientovi. Výsledná data můžeme také vizualizovat, pokud tento skript spustíme s volitelným argumentem *–debug*. Výsledek této vizualizace je vidět na obrázku 4.5

4.9.2 Neuronová síť MotionBERT

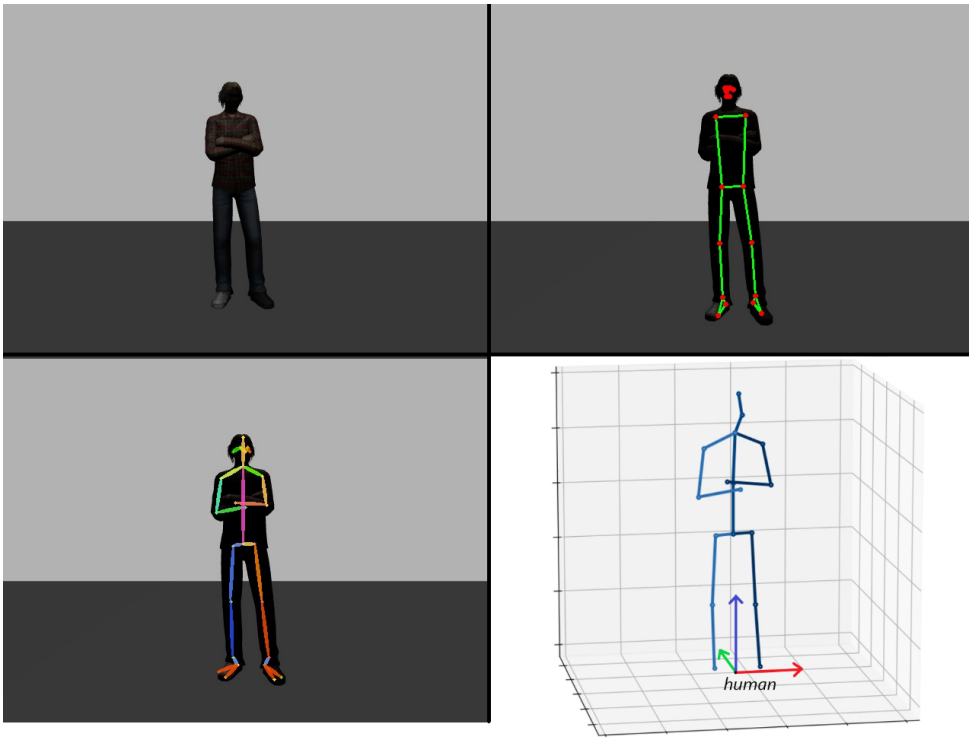
MotionBERT je předtrénovaný jednotný framework sloužící k vyřešení různých úkolů v oblasti analýzy lidského pohybu. Jedním z nástrojů tohoto frameworku je určení trojrozměrné pózy člověka s využitím předzpracovaných dat v AlphaPose. Výstupem jsou pak souřadnice 17 klíčových bodů na těle člověka. MotionBERT využívám v souboru `motionbert_image_server.py`, jehož základ pochází z ukázkových skriptů ve zdrojovém kódu tohoto frameworku na GitHubu [Wal23]. Zdrojový kód jsem si stáhl a několik souborů mírně poupravil tak, aby bylo možné zpracovávat data obdržená ze skriptu `main.py` a nebylo třeba je načítat z uložených souborů. Zároveň jsem soubor uzpůsobil k opakovanému rozpoznávání příchozích dat.

Popis souboru `motionbert_image_server.py`

Soubor je možné spustit s volitelným parametrem *–debug*, který zapne ukládání vizualizace na disk. Po jeho zapnutí se načte nastavení a poslední natrénovaný checkpoint pro detekci pózy člověka v obraze. Poté se inicializuje se serverová strana komunikačního kanálu a čeká se na data zpracovaná pomocí AlphaPose. Po jejich obdržení se data zpracují stejným způsobem jako v původních ukázkových programech. Výsledky rozpoznání se pošlou zpět do hlavního skriptu ve formě objektu typu `numpy.array`. Výsledek případné vizualizace v simulaci je zobrazený na obrázku 4.5 a ve skutečnosti pak na obr. 5.2.

4.10 Nalezeni finálního bodu

Po obdržení souřadnic klíčových bodů trojrozměrné kostry člověka můžeme přejít k nalezení místa, ne které člověk ukazuje pravou rukou. Tento proces je



Obrázek 4.5: Vizualizace procesu rozpoznávání pózy člověka v simulaci. Vlevo nahoře - záběr z kamery. Vpravo nahoře - výsledky rozpoznání pomocí MediaPipe. Vlevo dole - výsledky rozpoznání pomocí AlphaPose. Vpravo dole - Výsledky rozpoznání pomocí MotionBERT

realizován v `main.py` a byl rozdělen na několik menších podúkolů, kde každý z nich je realizován v jiné funkci.

Nejprve je potřeba obdržené body uspořádat do slovníku, abychom k jednotlivým bodům mohli přistupovat jménem dotyčného kloubu. Uspořádání je realizováno metodou `HumanInfo.create_joints_dict`.

Poté je nutné počátek souřadného systému nalezených bodů přesunout na zem mezi nohy člověka. Tento převod je realizován v metodě `HumanInfo.change_reference_of_joints` a výsledný souřadný systém budu nazývat *human*. Samotný výpočet bodu na podlaze podle teoretického postupu v sekci 3.4.2 je pak implementován ve funkci `get_point_on_ground`.

Nalezený bod je následně potřeba transformovat ze souřadného systému *human* do systému `odom`. Transformace jsou pak realizovány ve funkci `transform_points_to_odom`. Tím následně dostáváme souřadnice finálního bodu a můžeme je předat funkci `move_robot_to_final_point`. Ta pouze realizuje pohyb na nalezený finální bod a zarovnává hlavu robota.

4.11 Přehled vytvořených demo programů

Pro snadnější ověření fungování všech dílčích částí programu jsem vytvořil několik demo sekvencí, které jsou dostupné a spouštěné pomocí volitelného argumentu *–debug* při spouštění hlavního skriptu `main.py` jako "`-debug <nazev_dema>`"

Demo *camera*

Demo *camera* slouží k ověření správného fungování pořízení obrazu z kamery a jeho následného zpracování pomocí MediaPipe, případně pak AlphaPose a MotionBERT, pokud se v něm vyskytuje člověk. Ověří se tak zároveň i správné fungování komunikace mezi běžícími procesy.

Demo *robot_info*

Po spuštění tohoto příkazu se do terminálu vypíšou odometrická data robota, tedy jeho současné souřadnice a natočení základny robota v souřadném systému *odom* spolu s aktuálními rychlostmi.

Demo *move_head*

Toto demo ověří správné fungování ovládání pohybu hlavy robota. Sestává se ze sekvence natočení doleva, doprava, nahoru a dolů vždy o 30 stupňů.

Demo *rotate_base*

Během tohoto dema se základna robota otočí o 45 stupňů na obě strany od původního směru základny. Slouží k ověření plynulosti rotace základny.

Demo *move_forward_and_back*

Spuštěním tohoto dema robot popojede o jeden metr ve směru aktuálního otočení základny, následně se otočí a vrátí se zpět na původní místo. Demo slouží k ověření správného výpočtu transformace místa dojezdu do souřadného systému *odom* a zároveň k ověření přiměřeného nastavení rychlosti pohybu základny vpřed.

Demo *watch_human*

Během tohoto dema se robot snaží najít a udržet si člověka ve středu záběru ze své kamery. Slouží k otestování správného rozpoznání osoby v obraze a následné schopnosti a rychlosti reakce na pohyb člověka po pracovní ploše.

Demo *go_to_point*

Po spuštění tohoto dema má člověk 10 vteřin na postavení se do záběru robota, ten pak zpracuje pořízený snímek pomocí AlphaPose a MotionBERTa a na povel člověka provede přesun na nalezené místo, na které člověk ukazoval. Demo slouží k ověření přesnosti určení místa finálního dojezdu z gesta člověka.

4.12 Finální pracovní smyčka

Pokud byl skript `main.py` spuštěný bez volitelného argumentu *–demo*, spustí se hlavní finální smyčka programu. Na jejím začátku se zaregistruje skript jako *node* s názvem `tiago_python_controller` a inicializují se objekty tříd `TransformListener`, `BasePositionWithOrientation`, `ImageConverter`, `LatestJointStates`, `HumanInfo` a `Camera`, z nichž některé jsem popisoval v sekcích 4.5.1 a 4.5.2. Po jejich inicializaci se vytvoří komunikační klienti do ostatních skriptů po zpracovávání obrazu. Následuje vlastní pracovní smyčka. Pseudokód pro ní je zobrazen v algoritmu 4.

Na začátku si nastavím počítadlo zacentrovaných snímků a pokusů o nalezení člověka v záběru. Pak pokud po pořízení 3 snímků nenajdu člo-

věka v aktuální poloze, hledám člověka dle algoritmu `find_human`. Pokud člověka najdu, ale ještě není uprostřed záběru kamery, otočím na něj hlavou za pomocí funkce `center_camera_on_human`. Když se člověk udrží na jednom místě po 10 po sobě jdoucích kontrol obrazu z kamery a stále se nachází uprostřed záběru, spustí se rozpoznání obrazu pomocí AlphaPose (`analyze_picture_with_alphapose`), výsledky odešlu do MotionBERT (`analyze_picture_with_motionbert`), určím si vzdálenost člověka od kamery (`Camera.find_distance_to_human`), naleznu finální místo dojezdu (`find_final_point`) a robotu pošlu příkaz k přesunutí na finální místo. Po otočení do finálního směru robot opět hledá člověka a proce se opakuje.

Algoritmus 4: Hlavní pracovní smyčka

```
centered_iteratins ← 0;
tries ← 0;
while not rospy.is_shutdown() do
    if not human_info.is_found() & tries >= 3 then
        centered_iteratins ← 0;
        tries ← 0;
        find_human();
    else if
     human_info.is_found() & not human_info.is_centered()
     then
        centered_iteratins ← 0;
        tries ← 0;
        center_camera_on_human();
    else if not human_info.is_found() then
        centered_iteratins ← 0;
        tries ← tries + 1;
    else if
     human_info.is_found() & human_info.is_centered()
     then
        centered_iteratins ← centered_iterations + 1;
        tries ← 0;
        if centered_iteratins >= 10 then
            analyze_picture_with_alphapose();
            analyze_picture_with_motionbert();
            Camera.find_distance_to_human();
            final_point_matrix, θ ← find_final_point();
            move_robot_to_final_point(final_point_matrix, θ);
            centered_iterations ← 0;
        end
    end
    analyze_picture_with_mediapipe();
end
```

Kapitola 5

Výsledky a Diskuse

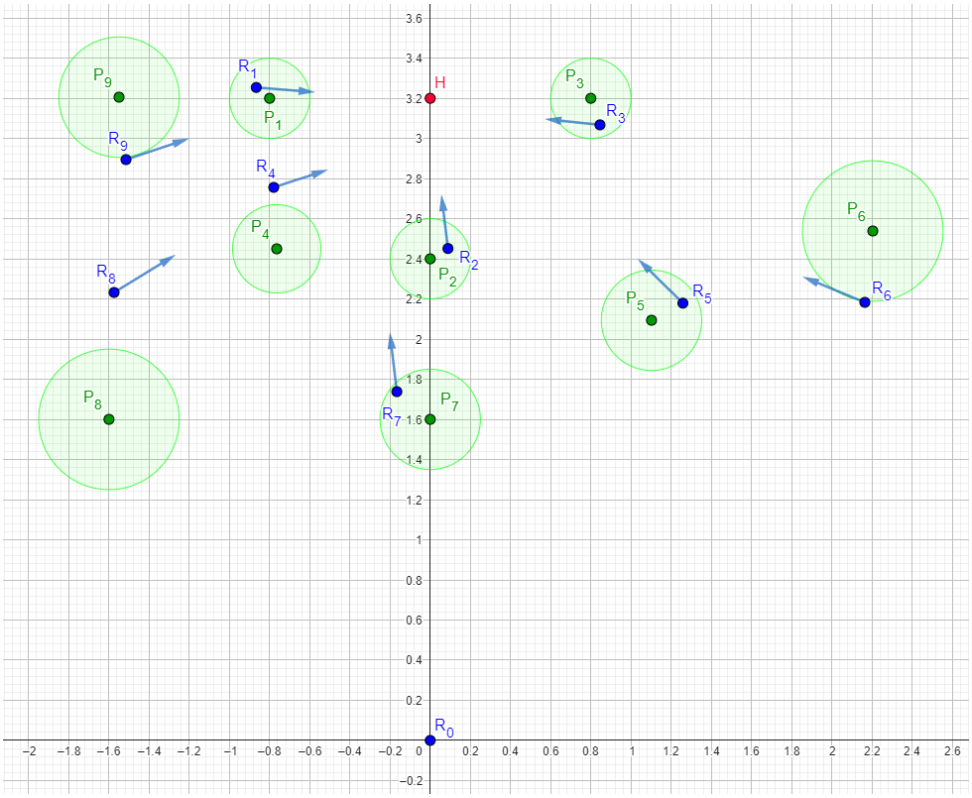
5.1 Sledování člověka

Robot je schopen najít člověka v libovolné části pracovního prostoru (pokud splňuje požadavky kladené na pracovní prostor v sekci 2.2) a následně si člověka udržet ve svém zorném poli, pokud se člověk po prostoru pohybuje dostatečně pomalu. Rychlost reakce na pohyb člověka je bohužel omezena rychlostí zpracování obrazu pomocí MediaPipe a rychlostí pohybu hlavy robota. Občas se stane, že robot chybně detekuje člověka v odrazu skla či při pohledu na robotické manipulátory v laboratoři. Člověka před sebou naopak nevidí, pokud je u robota příliš blízko nebo jsou v místnosti špatné světelné podmínky. Jinak celý systém sledování člověka a otáčení se za ním funguje správně.

5.2 Přesnost určení místa dojezdu

Přesnost rozpoznání místa dojezdu je složitá určit. První problém činí přesnost člověka při ukazování na chtěné místo. Proto je v grafu 5.1 znázorněna signalizovaná poloha dojezdu robota člověkem jako oblast a nikoliv jako jediný bod. Také platí, že čím vzdálenější indikovaný bod dojezdu od člověka je, tím větší bude oblast přibližného ukazování. Další potíž s přesností souvisí s využitou metodou zpracování obrazu. Aby zpracování obrazu bylo rychlé a

fungovalo v reálném čase, program pracuje pouze s jednou fotkou, ze které není schopen tak přesně určit prostorové souřadnice ruky. Proto jsou místa dojezdu pro robota omezena na blízké okolí kolem člověka, přibližně do vzdálenosti 1.5 metru. Při větších vzdálenostech již robot ztrácí schopnost přesně rozpoznat místo dojezdu. Přesnost jsem měřil opakovanými pokyny k pohybu z předem známé vzdálenosti do různých míst v pracovním prostoru mezi robotem a člověkem. Robot i člověk vždy stáli na stejných místech. Po dojezdu na místo se robot otáčí do směru, ve kterém se naposledy člověk nacházel. Před dalším pokusem jsem teleoperativně navedl robota do stejné počáteční polohy. Výsledky jsou graficky znázorněné na obr. 5.1



Obrázek 5.1: Vykreslení požadovaných oblastí dojezdu a finálních poloh středu základny robota (v metrech). Zelené kruhy - oblasti ukazovaného požadovaného dojezdu. Body R - Místa dojezdů robota a jeho finální orientace. R_0 - počáteční poloha robota. Bod H - poloha člověka.

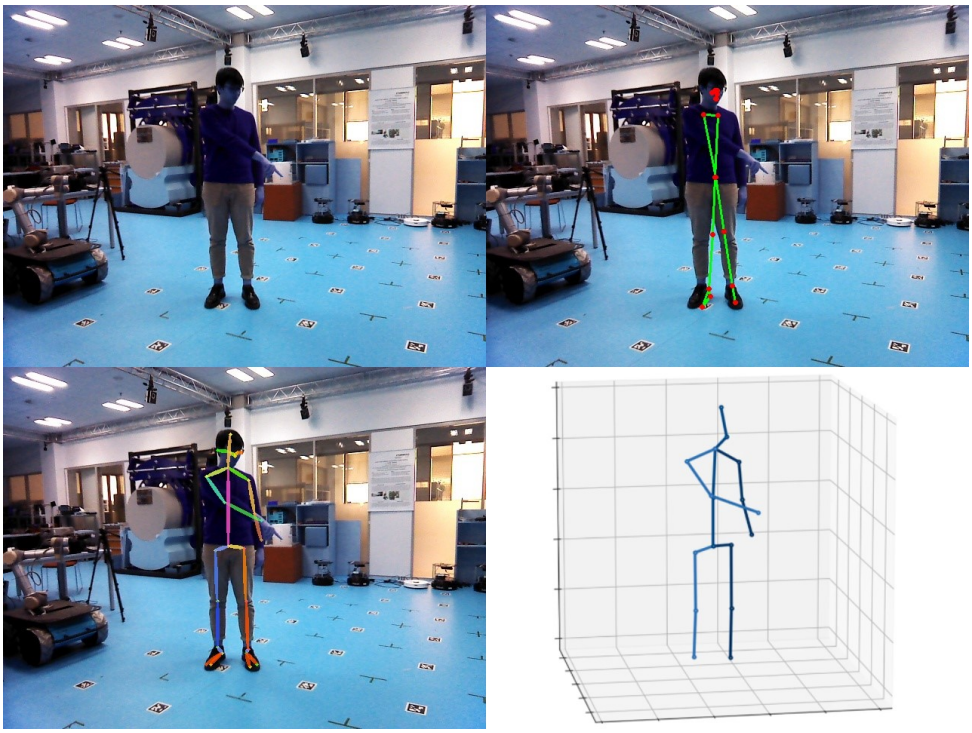
Z grafu je patrné, že s větší vzdáleností místa dojezdu od člověka klesá přesnost rozpoznání dojezdu robotem. Vektory z bodů dojezdu značí finální natočení robota po dojezdu do daného bodu. Při začátku pohybu robota se občas stalo, že se jedno z kol lehce protočilo a robot se tak natočil do trochu jiného směru, než do kterého měl jet. Odometrie však tuto chybu nemohla odhalit a ani tak směr robota korigovat.

Opakovatelnost příkazů a následná přesnost určení místa dojezdu byla

podobná jako při testování z pevně dané pozice. Při běhu hlavního programu je možné robotovi opakovaně zadávat příkazy po sobě bez nutnosti opakovaného spouštění programu či jiného přerušení. Robot tak byl schopen vykonat i 10 přesunů na požadované místo za sebou a udržel si i výše popsanou přesnost. Robot detekuje gesto tím způsobem, že kontroluje, zda se člověk nachází na stejném místě v obraze po 10 po sobě jdoucích snímků (přibližně 5-10 vteřin, záleží na rychlosti zpracování obrazu). Pokud se člověk po tuto dobu nepohnul, robot toto vnímá jako pokyn k pohybu a přejde ke vytváření 2D skeletonu.

Postup zpracování obrazu v simulaci byl již přiblížen na obrázku 4.5, Vizualizace reálných záběrů z robota je ukázána v obr. 5.2. Z něj je i vidět důvod, proč jsem pro vytvoření 2D skeletonu člověka nepoužíval znovu MediaPipe. Občas se stávalo, že byly pomocí něj rozpoznány jen nějaké z klíčových bodů. Kdežto pomocí AlphaPose byly vždy a spolehlivě rozpoznány všechny potřebné klíčové body.

Video demonstrující funkcionalitu řešení popsaného v této bakalářské práci je dostupné přes odkaz [Mik23]



Obrázek 5.2: Vizualizace zpracování zkutečného záběru z kamery robota. Vlevo nahoře - záběr z kamery. Vpravo nahoře - výsledky rozpoznání pomocí MediaPipe. Vlevo dole - výsledky rozpoznání pomocí AlphaPose. Vpravo dole - Výsledky rozpoznání pomocí MotionBERT

5.3 Návrhy na zlepšení

Celý systém rozpoznávání člověka a výpočtu místa dojezdu jistě není dokonalý. Určitě je možné zrychlit rozpoznávání člověka v obraze, například zkrácením doby pohybu hlavy, pokud je již člověk detekován.

Také je tu otázka zlepšení přesnosti. Ve své práci posílám ke zpracování pomocí AlphaPose a MotionBERT pouze jeden snímek, tyto nástroje jsou však primárně určené ke zpracovánání krátkých videí. Tato skutečnost mohla mít vliv na výslednou přesnost určování polohy pro dojezd robota.

Program je také zatím uzpůsobený pro čtení gest pouze pravé ruky. Modifikace na sledování i gest levé ruky je pak snadným vylepšením tohoto programu.

Dalším vylepšením by mohl být lepší systém signalizace gesta robotovi. Místo čekání na místě by se třeba mohl využít hlasový pokyn, po jehož zaznamenání by robot zahájil zpracování obrazu. Robot disponuje mikrofóny i reproduktory, tudíž by tato úprava byla možná.

Vzhledem k tomu, že nástroje pro zpracování obrazu nebyly přímo předučené na míru našemu zadání, byly výsledky s jejich použitím a implementací dle mého názoru dosti přesné. To dokazuje, že i s bezplatnými a volně dostupnými nástroji, které je člověk schopen vhodným způsobem spojit, je možnost dosáhnout zajímavých výsledků v oblasti humanoidní robotiky.

Příloha A

Literatura

[CPC22] Sungho Chun, Sungbum Park, and Ju Yong Chang, *Learnable human mesh triangulation for 3d human pose and shape estimation*, In Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (2022), pp. 2850–2859.

[Dyn23a] Boston Dynamics, *Atlas Gets a Grip | Boston Dynamics*, `https://www.youtube.com/watch?v=-e1_QhJ1EhQ`, January 2023, Online [navštíveno: 26. 5. 2023].

[Dyn23b] ———, *Atlas™*, `https://www.bostondynamics.com/atlas`, January 2023, Online [navštíveno: 26. 5. 2023].

[ET10] Mustafa Suphi Erden and Tetsuo Tomiyama, *Human-intent detection and physically interactive control of a robot without force sensors*, IEEE Transactions on Robotics **26** (2010), no. 2, 370–382.

[FLT+22] Hao-Shu Fang, Jiefeng Li, Hongyang Tang, Chao Xu, Haoyi Zhu, Yuliang Xiu, Yong-Lu Li, and Cewu Lu, *Alphapose: Whole-body regional multi-person pose estimation and tracking in real-time*, IEEE Transactions on Pattern Analysis and Machine Intelligence (2022).

[HS22] A. J. Hawkins and U. Shakir, *Tesla CEO Elon Musk unveils prototype humanoid Optimus robot*, `https://www.theverge.com/2022/9/30/23374729/tesla-bot-ai-day-robot-elon-musk-prototype-optimus-humanoid`, October 2022, Online [navštíveno: 26. 5. 2023].

[Hua17] Tony Huang, *How to move a robot to a certain point using twist*, `https://www.theconstructsim.com/`

ros-qa-053-how-to-move-a-robot-to-a-certain-point-using-twist/, November 2017, Online [navštíveno: 26. 5. 2023].

[Jal21] Marek Jalůvka, *Tiago++: a robotic archer*, Master's thesis, Czech Technical University in Prague, Prague, Czech Republic, May 2021.

[L.20] PAL Robotics S. L., *TIAGO++ Handbook*, 2020.

[LMR+15] Matthew Loper, Naureen Mahmood, Javier Romero, Gerard Pons-Moll, and Michael J. Black, *SMPL: A skinned multi-person linear model*, ACM Trans. Graphics (Proc. SIGGRAPH Asia) **34** (2015), no. 6, 248:1–248:16.

[LTN+19] Camillo Lugaresi, Jiuqiang Tang, Hadon Nash, Chris McClanahan, Esha Uboweja, Michael Hays, Fan Zhang, Chuo-Ling Chang, Ming Guang Yong, Juhyun Lee, Wan-Teh Chang, Wei Hua, Manfred Georg, and Matthias Grundmann, *Mediapipe: A framework for building perception pipelines*, arXiv 1906.08172 (2019).

[Mik23] MikeHubXP, *Gesture operation of robot tiago++*, https://youtu.be/sJkSQmx5dcA, May 2023, Online [navštíveno: 26. 5. 2023].

[MS23] MVIG-STJU, *Alphapose github repository*, https://github.com/MVIG-SJTU/AlphaPose, January 2023, Online [navštíveno: 26. 5. 2023].

[Noe16] Noel.martignoni, *Soubor:ros-master-node-topic.png*, https://fr.wikipedia.org/wiki/Fichier:ROS-master-node-topic.png#filelinks, January 2016, Online [navštíveno: 26. 5. 2023].

[OR] Open Robotics, *Gazeboo - Simulate before you build*, https://gazebosim.org/home, Online [navštíveno: 26. 5. 2023].

[OR21] ———, *Robot operating system*, https://www.ros.org/, 2021, Online [navštíveno: 26. 5. 2023].

[Pro23] Michal Procházka, *Bachelor thesis repository: Gesture operation of humanoid robot tiago*, https://github.com/MikeHubXP/bakalarka_tiago, May 2023, Online [navštíveno: 26. 5. 2023].

[RB22] PAL Robotics Blog, *TIAGo++, the robot you need for bi-manual tasks*, https://blog.pal-robotics.com/tiago-bi-manual-robot-research/, August 2022, Online [navštíveno: 26. 5. 2023].

[Rob23] Open Robotics, *Tiago++ tutorials*, http://wiki.ros.org/Robots/TIAGo%2B%2B/Tutorials, March 2023, Online [navštíveno: 26. 5. 2023].

[ros] *Robot operating system wiki*, http://wiki.ros.org/Documentation, Online [navštíveno: 26. 5. 2023].

[Thi10] ThibaultKruse, *Getting the current joint angles for the pr2*, `http://wiki.ros.org/pr2_controllers/Tutorials/Getting%20the%20current%20joint%20angles`, September 2010, Online [navštíveno: 26. 5. 2023].

[Wal23] Walter0807, *Motionbert github repository*, `https://github.com/Walter0807/MotionBERT`, May 2023, Online [navštíveno: 26. 5. 2023].

[ZML+22] Wentao Zhu, Xiaoxuan Ma, Zhaoyang Liu, Libin Liu, Wayne Wu, and Yizhou Wang, *Learning human motion representations: A unified perspective*, arXiv preprint arXiv:2210.06551 (2022).