

Institute of Architecture of Application Systems

University of Stuttgart  
Universitätsstraße 38  
D-70569 Stuttgart

Masterarbeit

# **Human Context-Awareness and its Application in a Service-Oriented Robotic System**

Vinayak Vishwas Patil

**Course of Study:** INFOTECH  
**Examiner:** Dr. Ilche Georgievski  
**Supervisor:** Nasiru Aboki, M.Sc.

**Commenced:** April 3, 2023  
**Completed:** October 3, 2023



## **Abstract**

As service robots become more prevalent in social settings, there is a growing imperative for research in the realm of social behavior development for these machines. An essential aspect of attaining autonomy in mobile service robots is their capacity to perceive and understand their surroundings effectively.

Despite being a formidable challenge, researchers have developed numerous solutions to empower robots with the ability to engage with and gain a deeper understanding of their environment. This enhanced capability endows robots with the intelligence required to make informed decisions by harnessing the knowledge they have accumulated.

In our research study, we have deployed a robotic system equipped with an autonomy solution that seamlessly incorporates human intentions via effective human-robot interactions. To be more precise, we utilize vision-based human pose estimation to identify gestures indicative of human intentions. Subsequently, these intentions are translated into predefined commands, facilitating the autonomous scheduling of tasks for the robotic system.

Our research provides a comprehensive overview of an autonomy solution tailored for mobile robots. It elaborates our approach to Gesture-based Robot Control and Navigation operations, leveraging human posture knowledge. Our key focus areas of emphasis encompass diverse techniques for acquiring knowledge, managing decisions, enhancing situational awareness, facilitating human-robot interaction, and strategic planning. Additionally, we present the findings derived from the experimental phase of physically implementing the robot.



# Contents

<b>1</b>	<b>Introduction</b>	<b>13</b>
1.1	Motivation . . . . .	15
1.2	Problem Statement . . . . .	15
1.3	Research Methodology . . . . .	16
1.4	Document Structure . . . . .	18
<b>2</b>	<b>Background Information</b>	<b>19</b>
2.1	Robot Operating System - ROS . . . . .	19
2.1.1	Introduction . . . . .	19
2.1.2	Concepts . . . . .	20
2.1.3	Client Libraries . . . . .	22
2.1.4	ROS commands . . . . .	22
2.2	Docker . . . . .	23
2.3	Ohmni Docker . . . . .	24
2.3.1	Creating a Robotics Application Using Robot Operating System (ROS) . . . . .	24
2.4	Openpose . . . . .	24
2.4.1	Introduction . . . . .	24
2.4.2	Method . . . . .	25
2.4.3	System . . . . .	28
2.4.4	Evaluation . . . . .	29
2.5	Lidar SLAM . . . . .	31
2.5.1	Operational principles of Lidar SLAM . . . . .	31
2.5.2	Hector SLAM . . . . .	32
2.6	Localization - Adaptive Monte Carlo Localization (AMCL) . . . . .	33
<b>3</b>	<b>State of the Art</b>	<b>35</b>
3.1	Environmental Awareness in Mobile Service Robots . . . . .	35
3.2	Elevating Human-Robot Interaction . . . . .	35
3.3	Human-Robot Synergy for Enhanced Exploration . . . . .	36
<b>4</b>	<b>Design of Solution</b>	<b>39</b>
4.1	Hardware Setup . . . . .	39
4.1.1	Microsoft Kinect RGB-D Camera v1 . . . . .	39
4.1.2	YD LiDAR X4 . . . . .	40
4.1.3	System Hardware Specifications . . . . .	41
4.2	Towards seamless Human-Robot Interaction (HRI) . . . . .	41
4.2.1	Perception . . . . .	42
4.3	Gesture and Posture Detection . . . . .	44
4.3.1	Gesture Estimation from Pose Implementation . . . . .	44

4.3.2	Posture Estimation from Pose Implementation . . . . .	51
4.3.3	Custom ROS message . . . . .	56
4.4	Social Rule's . . . . .	56
4.5	Autonomous Navigation . . . . .	57
4.5.1	Environment Mapping - Hector SLAM . . . . .	57
4.5.2	Navigation Stack - Butler . . . . .	60
<b>5</b>	<b>Realization of Solution</b>	<b>67</b>
5.1	ROS Driver for Kinect Depth Camera . . . . .	67
5.1.1	Camera Calibration . . . . .	68
5.2	Docker Image - Openpose Library . . . . .	68
5.2.1	NVIDIA Container Toolkit and Configuring Docker. . . . .	68
5.2.2	Openpose Docker file . . . . .	69
5.2.3	ROS - Openpose v1.7.0 . . . . .	71
5.3	Communication with Butler . . . . .	73
5.4	Gesture Navigation - Butler . . . . .	76
5.4.1	Efficient Methodical Approach to Environment Mapping . . . . .	76
5.4.2	Effective Navigation Stack Implementation . . . . .	78
5.5	Time synchronization on Multiple Machine's . . . . .	82
<b>6</b>	<b>Evaluation of Solution</b>	<b>85</b>
6.1	Assessing Posture and Gesture Accuracy for Robot Control . . . . .	85
6.2	Assessing Posture and Gesture Accuracy for Robot Navigation . . . . .	86
6.3	Enhancing Robot Performance: Balancing GPU Utilization and Computational Overheads . . . . .	86
<b>7</b>	<b>Conclusion</b>	<b>89</b>
7.1	Summary . . . . .	89
7.2	Future Work . . . . .	90
	<b>Bibliography</b>	<b>91</b>

## List of Figures

1.1	Butler Robot . . . . .	14
2.1	Overall pipeline. . . . .	25
2.2	Architecture of the multi-stage CNN [7]. . . . .	26
2.3	Part association strategies.[7]. . . . .	28
2.4	Inference time comparison between OpenPose, Mask R-CNN, and Alpha-Pose [7].	29
2.5	Trade-off between speed and accuracy for the main entries of the COCO Challenge [7]. . . . .	30
4.1	Kinect RGB Camera [60] . . . . .	39
4.2	Kinect IR Camera [60] . . . . .	39
4.3	Kinect IR Projector [60] . . . . .	39
4.4	YDLIDAR X4 Development Kit . . . . .	41
4.5	Software Architecture Overview . . . . .	42
4.6	Output from OpenPose. . . . .	44
4.7	Obey Gesture . . . . .	46
4.8	Disobey Gesture . . . . .	47
4.9	Go forward gesture with left hand . . . . .	47
4.10	Go forward gesture with right hand . . . . .	47
4.11	Go backward gesture . . . . .	48
4.12	Go right gesture . . . . .	49
4.13	Go left gesture . . . . .	49
4.14	Go forward with right turn gesture . . . . .	50
4.15	Go forward with left turn gesture . . . . .	50
4.16	Facing towards Robot . . . . .	52
4.17	Facing backwards w.r.t Robot . . . . .	52
4.18	Facing sideways 1 <sup>st</sup> Posture . . . . .	53
4.19	Facing sideways 2 <sup>nd</sup> Posture . . . . .	53
4.20	First Criteria : Representation of $\theta$ . . . . .	54
4.21	Sitting 1 <sup>st</sup> Posture . . . . .	54
4.22	Sitting 2 <sup>nd</sup> Posture . . . . .	54
4.23	Sitting 3 <sup>rd</sup> Posture . . . . .	55
4.24	Ohmni Robot Unified Robotic Description Format (URDF) file. . . . .	58
4.25	Position of Light Detection and Ranging (LiDAR) in URDF file. . . . .	58
4.26	YDLIDAR X4 coordinates definition . . . . .	59
4.27	Map generated by Hector SLAM . . . . .	60
4.28	Navigation Stack Setup [ <a href="#">ROS Wiki</a> ] [33] . . . . .	62
4.29	ohmni_bring_up.launch file . . . . .	62
4.30	Common configuration file . . . . .	63

4.31	Global costmap parameters configuration file . . . . .	63
4.32	Local costmap parameters configuration file . . . . .	64
4.33	DWA Local Planner parameters configuration file . . . . .	65
4.34	Move Base parameters configuration file . . . . .	65
4.35	AMCL configuration parameters . . . . .	66
4.36	Launch File for the Navigation Stack . . . . .	66
5.1	Openpose Docker file . . . . .	69
5.2	config_kinectxbox_launch file . . . . .	72
5.3	Hector SLAM launch file . . . . .	77
5.4	Navigation Map Visualization . . . . .	78
5.5	Obey Gesture . . . . .	79
5.6	Disobey Gesture . . . . .	79
5.7	Go to Location A . . . . .	79
5.8	Go to Location B . . . . .	79

## List of Tables

4.1	PostureGesture.msg ROS Custom Message Structure . . . . .	56
5.1	Frame.msg ROS Custom Message Structure . . . . .	73
6.1	Gesture Detection Results . . . . .	85
6.2	Posture Detection Results . . . . .	85
6.3	Robot Position Deviation and Error Percentages . . . . .	86

## List of Algorithms

5.1	Gestures-based Control Algorithm . . . . .	74
5.2	Posture and Facing Detection Algorithm . . . . .	75
5.3	Gestures-based Navigation Algorithm . . . . .	81



# Acronyms

<b>ADB</b>	Android Debug Bridge.	73
<b>AI</b>	Artificial Intelligence.	15
<b>AMCL</b>	Adaptive Monte Carlo Localization.	5
<b>CPU</b>	Central Processing Unit.	86
<b>FPS</b>	Frames Per Second.	89
<b>GPU</b>	Graphics Processing Unit.	86
<b>GUI</b>	Graphical User Interface.	78
<b>HMI</b>	Human-Machine Interaction.	13
<b>HRC</b>	Human-Robot Collaboration.	80
<b>HRI</b>	Human-Robot Interaction.	5
<b>IAAS</b>	Institute for Architecture of Application Systems.	13
<b>LAN</b>	Local Area Network.	82
<b>LiDAR</b>	Light Detection and Ranging.	7
<b>NTP</b>	Network Time Protocol.	82
<b>PAFs</b>	Part Affinity Fields.	25
<b>PGM</b>	Portable Gray Map.	77
<b>ROS</b>	Robot Operating System.	5
<b>RVIZ</b>	ROS Visualization.	77
<b>SLAM</b>	Simultaneous Localization and Mapping.	16
<b>URDF</b>	Unified Robotic Description Format.	7
<b>WRI</b>	Worker-Robot Interaction.	13
<b>YAML</b>	Yet Another Markup Language.	77



# Acknowledgements

The sense of satisfaction and euphoria that accompanies the successful completion of any endeavor would remain incomplete without acknowledging the individuals who played a pivotal role in making it possible. Their unwavering guidance and encouragement have been instrumental in bestowing success upon all the diligent efforts.

I am sincerely and profoundly grateful to Prof. Dr. Marco Aiello for granting me the opportunity to collaborate on Research Project involving Ohmni Telepresence Robot. This experience marked the beginning of my fascinating voyage into the realm of Robotics, and for that, I am truly indebted.

I extend my heartfelt gratitude to my supervisor, Mr. Nasiru Aboki, for his continuous support, invaluable guidance, insightful advice, and relentless encouragement throughout the entire journey of my thesis work. His unwavering commitment has elevated the quality of my work to new heights.

Furthermore, I would like to express my deep appreciation to Dr. Ilche Georgievski for meticulously examining and reviewing my thesis work at every step of the process.

Last but certainly not least, I wish to extend my heartfelt gratitude to my family and my dearest friend, Krishna, for their unwavering moral support and encouragement throughout the entirety of my Master's studies.



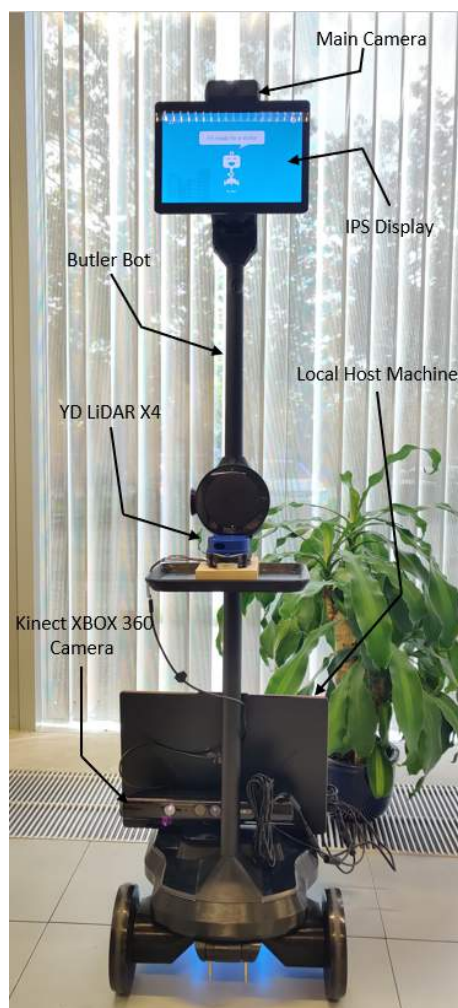
# 1 Introduction

The concept of our society transitioning into a “robot society” is gaining traction. The New York Times’ video series “Robotica” further showcased the diverse applications of robots in daily life, shedding light on both their potential and limitations, thereby emphasizing the necessity for a deeper comprehension. This growing public interest is paralleled by an increasing number of policy reports and scientific articles, indicating the imminent integration of robots into labor and everyday life. As early as 1996, Japanese scientist Masakazu Ejiri delved into the future of robotics [31]. He asserted that many experts in the field believe autonomous robots will wield significant influence in our forthcoming society. Ejiri, associated with the renowned Hitachi Lab, specifically focused on scenarios where robots are prevalent in work settings. He proposed that forthcoming technical advancements in robot mobility, motion control, energy storage, and battery technology will largely address the challenges associated with robots.

The “social” dimensions concept arises from the interaction between workers and robots in industrial settings, encompassing aspects of safety, control systems, and interface ergonomics. These factors are becoming increasingly vital not only in the realm of Worker-Robot Interaction (WRI) but also in the broader context of HRI. HRI is applicable across various domains, including service robotics and manufacturing processes. Within this framework, HRI encompasses traditional elements of Human-Machine Interaction (HMI), such as communication with machines, mediation between humans and objects, the necessity for anticipation, simulation, and more. Consequently, there is a growing recognition among system integrators, robot manufacturers, and companies about the significance of considering “social” aspects beyond just ergonomic and safety concerns. The term “social” dimensions here encompasses not only physical factors but also new qualifications, technical competences, and organizational skills. These encompass working team dynamics, communication proficiencies, and decision-making processes within workflows. These considerations are pivotal when designing systems that facilitate interactions between humans and robots. Addressing these aspects often requires a sociological perspective [31].

Beyond serving as a mere remote-controlled device, telepresence robots hold a more profound significance. The Institute for Architecture of Application Systems (IAAS) laboratory features a cutting-edge [telepresence robot](#) [38] specifically designed by OhmniLabs, a Silicon Valley company established in 2015. The particular variant in use revolutionizes interpersonal connections by virtually placing individuals in remote settings. The Ohmni Telepresence Robot, which we refer to as the ‘Butler’ 1.1, this robot is exceptionally suited for office environments. Operating as a mobile wheeled device, the telepresence robot is equipped with wireless internet connectivity. It can be easily maneuvered using a computer, tablet, or smartphone. This interaction facilitates a comprehensive exchange between users and the robot operator by enabling them to simultaneously observe and hear the operator via their display screen. We leverage the developer edition of the Ohmni Telepresence Robot, which empowers users to enhance and extend its functionalities.

This thesis is dedicated to exploring the domain of robotic visual perception, with the overarching objective of designing a comprehensive robot architecture. This architecture aims to empower the robot not only to achieve high-level goals but also to navigate the complexities of its dynamic environment by acquiring and processing information. The central ambition of this work revolves around creating an intelligent mobile robot capable of deciphering the intricate semantics inherent in human environments. This involves a deep understanding of the spatial relationships that exist between humans and their surroundings. This, in turn, leads to the development of a heightened sense of robotic awareness. To realize this ambition, we have selected the ROS as our middleware framework. This choice affords us the advantages of a publish-subscribe communication paradigm, alongside the incorporation of services and actions. Moreover, ROS provides a robust mechanism for managing concurrent processes or nodes, enabling seamless coordination within the system.



**Figure 1.1:** Butler Robot

## 1.1 Motivation

Endowing service robots with the capacity to comprehend their surroundings stands as a pivotal element, particularly in settings focused on human presence like residences and workplaces. The ultimate objective of a service robot resides in acquiring, comprehending, and proficiently carrying out tasks that might pose challenges for humans. Typically, individuals undertake actions grounded in their perception of circumstances and the environment, leveraging their acquired knowledge pertaining to the assorted elements within the surroundings. While this cognitive process comes instinctively to humans, robots lack the inherent capacity required for decision-making and action execution. Consequently, the integration of environmental awareness and situational understanding becomes imperative for robots. The acquisition of knowledge can be delineated as a robot's capability to gather and retain information regarding its operational world. Planning and reasoning systems are harnessed to process this acquired knowledge, enabling robots to proficiently execute tasks.

The environment encompasses the entirety of the area within which a robot functions, encompassing all the entities situated therein. A situation is a limited segment of the environment, relevant to a specific task and time-frame. Equipping robots to acquire knowledge from interactions and grasp semantic relations helps them understand various situations and respond suitably. However, this is intricate, as robots can't naturally introspect. Artificial Intelligence (AI) offers tools, yet with constraints. Also, robots must prioritize human safety, interpret intentions, and anticipate outcomes. These complexities underscore the challenge of knowledge acquisition and reasoning in service robotics. Contemporary approaches often assume the human brain holds either innate knowledge triggered by stimuli or a mechanism to gather knowledge from related experiences. In service robotics, knowledge systems are segregated into domains like encyclopedic, common sense, and domain-specific knowledge. The robot's surroundings encompass diverse entities physical objects, human actions, articulated movements, among others. Acquiring these knowledge types is vital for robots to emulate human capabilities [42].

This Thesis aims to enhance the Ohmni telepresence robot, enabling it to effectively engage and assist people in their daily routines. The focus goes beyond conventional robotics aspects like motion planning and navigation, encompassing human interaction abilities. Notably, the project centers on telepresence robot adept at coexisting with humans, offering diverse levels of interaction with one or multiple users.

## 1.2 Problem Statement

As discussed in preceding sections, a significant contemporary robotics challenge revolves around the development of service robots capable of accomplishing diverse tasks while in the company of humans. The principal goal of this thesis is to devise a robot framework that facilitates the attainment of advanced objectives amidst complex and ever-shifting surroundings. Specifically, the Ohmni telepresence robot at the Service Computing Lab of IAAS department must proficiently engage in tasks such as environmental perception, navigation, and human interaction, all in a simultaneous manner. This entails the creation of specialized structures adept at harmoniously orchestrating multiple tasks, often operating in parallel and seamless coordination.

Building upon the previously stated assumption, this research addresses to tackle the following questions:

- **Research Question 1:** How does the incorporation of vision-based perception into robotics play a critical role, and to what extent can camera sensors effectively meet the demands of tasks such as autonomous navigation, human-robot interaction, and collaboration?
- **Research Question 2:** What range of decisions can an autonomous robot effectively undertake through the utilization of camera sensors, which are pivotal in vision-guided systems, and what distinct benefits arise from the decisions enabled by a camera endowed with depth and motion sensing capabilities?
- **Research Question 3:** How do you propose to enable an autonomous robot to perform real-time detection, recognition, and classification of human skeletal frames, while also conducting pose estimation through knowledge-based reasoning, especially when faced with such scenarios while pursuing its objectives?

### 1.3 Research Methodology

The research endeavor aims to investigate the application of the Openpose software framework in conjunction with a non-odometric approach utilizing Hector Simultaneous Localization and Mapping (SLAM) for path planning and navigation, within the ROS framework. This study represents a notable progression within the field of HRI and perception, offering the prospect of significant advancements in this domain. The core focus of this research is dedicated to the comprehensive assessment of outcomes and the meticulous execution of a performance analysis, which pertains to the development and seamless integration of a diverse suite of modules. These encompass the critical domains of image analysis, decision-making processes, HRI capabilities, and autonomous navigation systems, all of which collectively constitute the essential facets of this multifaceted inquiry.

The subsequent methodology was employed to address the aforementioned research inquiries:

- In the realm of real-time 2D pose estimation, a substantial body of literature and research exists, reflecting its prominence in computer vision and robotics. Consequently, it becomes imperative to leverage the knowledge and insights from prior research endeavors that have employed 2D pose estimation in analogous scenarios or with similar objectives.

Similarly, in the domain of robotics and autonomous navigation, Simultaneous Localization and Mapping (SLAM) has achieved significant milestones and attracted extensive research attention. It is essential to delve into and harness insights and methodologies from previous research efforts that have employed SLAM on differential drive robots or in analogous environments.

This strategic utilization of existing research not only facilitates the adoption of a more suitable and informed methodology but also offers valuable insights into the strengths and limitations of current research within this specific domain. This, in turn, enhances the credibility and relevance of our ongoing work.



- Our formal work began with a thorough investigation into two essential areas: the ROS Framework and Docker technology. Notably, the Ohmni Developer Edition includes a powerful Docker virtualization layer, which is a valuable tool for ROS developers. This allows ROS developers to easily use the ROS framework on the Ohmni Operating System. This combination provides an effective platform for deploying and using ROS applications within the Ohmni ecosystem [Ohmni Developer](#) [39].
- The process of system design is a fundamental and critical aspect of development work, as it involves the meticulous definition of both hardware and software components. A system, in this context, represents a sophisticated framework characterized by the organized collaboration of various component groups, all united in the pursuit of a common goal. Thoroughly breaking down the system into its individual parts is a crucial step in revealing the complex connections and interdependencies among these components. Furthermore, this analytical approach sheds light on potential failure points that may materialize should certain constituent parts cease to function. Consequently, this systematic breakdown serves as a means to enhance work quality and substantially mitigate the occurrence of errors.
- Following the development of the system architecture, the next critical step involved setting up the necessary hardware components. It was essential to meticulously evaluate the needs of the tasks earmarked for computerization, encompassing the selection of an appropriate operating system to support development, the choice of a middleware framework for algorithm execution, and the establishment of a virtualization environment to enhance simulation efficiency.
- After finalizing the architectural framework, the subsequent phase entailed the development of the software program and the meticulous selection of appropriate ROS algorithms and software packages for each specific use case. An imperative aspect of this process involved ensuring seamless integration between the newly introduced software components and the pre-existing ones already installed within the system.
- During the course of our developmental process, we adopted a systematic approach in which we decomposed the primary features into smaller, manageable tasks, addressing each use case sequentially. Initially, our focus was on the successful realization of a Custom Docker Image for Openpose. Subsequently, we advanced to the task of enabling the control of a telepresence robot through gestures, leveraging the outcomes of the previous phase.

Following this, our attention turned to the development of an algorithm capable of accurately determining human posture, a critical component in our project. Concurrently, we laid the groundwork for the integration of Gesture and Navigation functionalities, necessitating the creation of an environmental map, the configuration of the robot's navigation stack, and the enhancement of dynamic obstacle avoidance capabilities, surpassing the effectiveness of prior implementations.

Ultimately, our efforts culminated in the successful achievement of Gesture-controlled Navigation for the telepresence robot, thereby synergizing our diverse use cases and achieving a comprehensive and cohesive solution.

## 1.4 Document Structure

The document is organized into several chapters as follows:

- Chapter 2 provides essential background information on key robotics technologies. It introduces the ROS and Docker, discusses Openpose, Lidar SLAM principles, and AMCL. This foundation sets the stage for deeper discussions in subsequent chapters.
- Chapter 3 explains the state-of-the-art technologies and the existing relevant work related to this thesis. Further, based on the evaluation, three specific research questions are formulated concerning Human Context-Awareness and its Application in a Service-Oriented Robotic System.
- Chapter 4 primarily focuses on delineating the hardware setup designed for the gesture and posture detection solution, incorporating social rules. Additionally, it presents a comprehensive design solution for enabling autonomous navigation of Butler Bot through gestures.
- Chapter 5 thoroughly explores the implementation of the solution, providing a detailed explanation of the process involved in communicating with Butler Bot and effectively executing Gesture-Based Bot Control. This chapter also delves into Gesture-Based Bot Navigation, where the system possesses an understanding of human posture, all while ensuring that machine timing remains synchronized with each other.
- Chapter 6 centers on evaluating the solution's performance, assessing posture and gesture accuracy for robot control and navigation, and addressing the optimization of robot performance by managing GPU utilization and computational overheads. This chapter is pivotal in determining the practicality and efficiency of the solution.
- Chapter 7 serves as the conclusion of the work and outlines potential avenues for future research and development.

## 2 Background Information

### 2.1 Robot Operating System - ROS

#### 2.1.1 Introduction

- **What is ROS?**

ROS, which stands for Robot Operating System, is an open-source and versatile meta-operating system designed to cater to the intricate needs of robotic systems. This sophisticated platform offers a wide array of services akin to those provided by conventional operating systems. These encompass hardware abstraction, precise control of low-level devices, the realization of frequently used functionalities, seamless message exchange among processes, and efficient package management. Additionally, ROS equips developers with a rich set of tools and libraries essential for acquiring, compiling, authoring, and executing code seamlessly across multiple computational nodes. ROS, as a system, does not inherently operate in real-time. Nevertheless, it is conceivable to integrate ROS with real-time code to achieve temporal precision and responsiveness.

- **Goals**

The primary objective of ROS is to facilitate the reutilization of code within the realm of robotics research and development. ROS embodies a distributed framework composed of processes, also referred to as Nodes, which afford the capability for individual executables to be meticulously crafted and loosely interconnected during runtime. These processes can be logically organized into Packages and Stacks, thereby simplifying the sharing and dissemination of robotic software components.

ROS is deliberately engineered to maintain a minimalistic footprint, refraining from encapsulating the `main()` function, thereby enabling seamless integration of code developed for ROS with other robot software frameworks.

Moreover, ROS advocates the development of ROS-agnostic libraries with well-defined and clean functional interfaces, thus promoting modularity and code reusability. Language agnosticism is another prominent feature of ROS, as it is designed to be effortlessly implemented in various modern programming languages. This versatility is evidenced by existing implementations in Python, C++, and Lisp, with ongoing experimental endeavors to introduce libraries in Java and Lua.

- **Operating Systems**

ROS is presently exclusively compatible with Unix-based operating systems. The software designed for ROS is predominantly validated on Ubuntu and Mac OS X environments, although the ROS community actively engages in efforts to extend support to additional Linux platforms such as Fedora, Gentoo, and Arch Linux.

It is worth noting that while the prospect of porting ROS to Microsoft Windows remains a possibility, it remains an area that has not been comprehensively investigated or implemented at this juncture [55].

### 2.1.2 Concepts

- **ROS Filesystem Level**

Filesystem-level concepts in the Robot Operating System (ROS) primarily encompass various resources encountered on the system's disk. These resources serve as fundamental components within ROS, facilitating the organization and management of robotic software. Key elements within this framework include:

1. **Packages:** At the core of ROS's software organization, packages are the fundamental building blocks. Each package can encompass a range of components, such as ROS runtime processes (referred to as nodes), ROS-dependent libraries, datasets, configuration files, or any other elements that are logically grouped together. Packages are the smallest, indivisible units for both building and releasing software in ROS. Consequently, packages represent the most granular entities that can be constructed and distributed.
2. **Metapackages:** Metapackages serve as specialized packages designed to group and represent collections of related packages. Typically, metapackages are employed as placeholders for maintaining backward compatibility when transitioning from the older rosbuilt Stacks system. They facilitate the management of interconnected packages within ROS.
3. **Repositories:** Repositories are repositories of packages that share a common Version Control System (VCS). Packages within the same repository are typically maintained at the same version and can be collectively released using automated tools like catkin's release automation tool, bloom. These repositories often align with converted rosbuilt Stacks, and it is worth noting that a repository can house either a single package or multiple related ones.
4. **Message (msg) Types:** Message types play a crucial role in defining the data structures used for messages exchanged within ROS. These descriptions are stored in specific directories, such as `my_package/msg/MyMessageType.msg`. Message types facilitate the communication of data between different components of a robotic system, allowing for the transmission of information in a standardized format.

5. **Service (srv) Types:** Service types, `my_package/srv/MyServiceType.srv`, define the structures for requests and responses in ROS services. These descriptions outline the data formats and parameters required when interacting with ROS services, enabling the seamless exchange of information between nodes in a robotic system.

- **ROS Computation Graph Level**

The Computation Graph in ROS represents a peer-to-peer network of interconnected ROS processes collaborating to process data. Within this framework, several fundamental concepts play crucial roles in facilitating data exchange and computation. These core Computation Graph concepts are implemented within the `ros_comm` repository. They include:

1. **Nodes:** Nodes are individual processes responsible for carrying out computation tasks. ROS embraces a modular approach, and a typical robot control system comprises multiple nodes. These nodes have diverse functionalities, such as controlling sensors (e.g., laser range-finders), managing actuators (e.g., wheel motors), executing localization and path planning, offering graphical system views, and more. Nodes are created using ROS client libraries like `roscpp` or `rospy`.
2. **Master:** The ROS Master serves as a vital component, providing name registration and lookup services to the entire Computation Graph. It is the linchpin that enables nodes to discover one another, exchange messages, and invoke services. Without the Master, the collaborative network of nodes would not function.
3. **Parameter Server:** The Parameter Server serves as a centralized repository for storing data, which can be accessed via unique keys. Currently integrated with the Master, it offers a convenient means to manage and share configuration parameters across nodes.
4. **Messages:** Communication between nodes occurs through the exchange of messages. Messages are essentially structured data entities, consisting of typed fields. They support standard primitive data types (e.g., integers, floating-point numbers, booleans) and arrays of these types. Messages can also include nested structures and arrays, much like C structs, enabling flexible data representation.
5. **Topics:** Messages are transmitted within the Computation Graph using a publish-subscribe mechanism. A node sends a message by publishing it to a specific topic, identified by a name. Nodes interested in particular data subscribe to relevant topics. Multiple publishers and subscribers can coexist for a single topic, and nodes can engage with multiple topics simultaneously. This design decouples data production from consumption, forming a type-safe message bus.
6. **Services:** While the publish-subscribe model suits many scenarios, it doesn't align well with request-reply interactions essential in distributed systems. Services address this need, utilizing pairs of message structures: one for the request and one for the reply. Nodes offering services expose them under specific names, while clients use these services by sending requests and awaiting responses. ROS client libraries often present this interaction in a manner akin to remote procedure calls (RPC). [55]

### 2.1.3 Client Libraries

A ROS client library constitutes a comprehensive codebase that simplifies the tasks of ROS programmers by providing programmatic access to key ROS concepts. These libraries abstract complex ROS functionalities into programming constructs, enabling developers to seamlessly create ROS nodes, engage in topic publication and subscription, implement and invoke services, and utilize the Parameter Server. While these libraries can be implemented in various programming languages, the current emphasis is on delivering robust support for both C++ and Python. Specifically, two prominent ROS client libraries are noteworthy:

1. **roscpp**: roscpp stands as the C++ client library of choice within the ROS ecosystem. It holds the distinction of being the most widely adopted ROS client library, designed with a focus on high-performance operation. This library empowers developers to harness the full capabilities of ROS through C++ code, making it ideal for resource-intensive ROS applications.
2. **rospy**: rospy, on the other hand, is the Python-based client library for ROS. It has been crafted to leverage the advantages of an object-oriented scripting language in the context of ROS. rospy's design prioritizes rapid implementation and testing of algorithms, making it an excellent choice for quickly prototyping ROS applications and non-critical code sections. Additionally, rospy's type introspection capabilities are beneficial for a range of tasks. Notably, several essential ROS tools, including the ROS Master and roslaunch, are developed using rospy, underscoring Python as a core dependency in the ROS ecosystem. [55]

### 2.1.4 ROS commands

Below is a list of commonly employed command-line tools for orchestrating ROS within the context of our research endeavors:

- **roscore**: This tool constitutes an ensemble of nodes and programs that are indispensable for the seamless operation of a ROS-based system. Upon execution, it initializes critical components including the ROS master, ROS parameter server, and the roscore logging node.
- **roslaunch**: Roslaunch is an invaluable utility designed for the systematic initiation of a collection of nodes, sparing the necessity of manual invocation for each node. It operates on '.launch' files, which encapsulate configurations of nodes and facilitate topic remapping.
- **rosvrun**: Rosvrun provides an expedient method for locating and executing packages within the file system. Notably, it obviates the need for explicit knowledge of the package path, streamlining the execution process.
- **rostopic**: This tool furnishes the capability to inspect and display messages transmitted across ROS topics, as well as furnish real-time insights into the status and dynamics of topics.
- **rostopic**: Rosnode serves as a diagnostic tool, offering detailed insights into ROS nodes. It provides information pertaining to node connections, publications, and subscriptions, which is indispensable for debugging and analysis.

The [ROS cheatsheet](#)'s provenance encompasses a comprehensive array of ROS commands [50].

## 2.2 Docker

Docker represents a suite of Platform as a Service (PaaS) offerings that leverage the concept of Operating System-level virtualization to facilitate the deployment of software within encapsulated units referred to as containers. These containers are characterized by their inherent isolation from one another and their self-contained provisions of software, libraries, and configuration files. Furthermore, containers establish intercommunication through clearly defined channels. Notably, all containers are executed under a unified operating system kernel, thereby achieving resource efficiency superior to that of virtual machines [Docker Introduction] [14].

### Important Terminologies in Docker

1. **Docker Image:** A Docker image is a composite entity composed of multiple layers, serving as the blueprint for the execution of code within a Docker container. Docker images consist of a series of instructions that are employed to instantiate Docker containers.
2. **Docker Container:** A Docker container is a live runtime instance derived from a Docker image. These containers empower developers to encapsulate applications along with their requisite components, encompassing libraries and dependencies.
3. **Dockerfile:** A Dockerfile represents a textual document housing imperative commands that, upon execution, facilitate the construction of a Docker Image. Docker images are synthesized using the directives prescribed within a Dockerfile.
4. **Docker Engine:** Docker Engine denotes the foundational software responsible for orchestrating and managing containers. This software operates within a client-server paradigm, comprising three primary constituents:
  - **Server:** The server component, referred to as a daemon process, undertakes the role of generating and overseeing Docker images, containers, networks, and storage volumes.
  - **REST API:** The REST API outlines the interface through which external applications can interact with the Docker Server, issuing instructions for various operations.
  - **Client:** The Client manifests as the Docker command-line interface (CLI), serving as a conduit for users to engage with Docker through a repertoire of command-driven actions.
5. **Docker Hub:** Docker Hub stands as the official online repository where an extensive catalog of Docker Images is hosted and made available for utilization. It streamlines the process of discovering, managing, and sharing container images, enhancing collaboration and accessibility among the Docker community.

### 2.3 Ohmni Docker

The Ohmni Developer Edition is equipped with a robust Docker virtualization layer, enabling the execution of various Ubuntu versions within the Ohmni platform.

By default, the command 'dockerenv' can be executed from either an adb or ssh shell. This command initiates the retrieval of the 'ohmnilabs/ohmnidev' image from our Docker Hub repository. This image serves as a foundation, featuring Ubuntu 18.04 as its base, supplemented with a selection of additional tools [[Ohmni Docker - Overview](#)] [36].

#### 2.3.1 Creating a Robotics Application Using ROS

The Ohmni Developer Edition incorporates a robust Docker virtualization layer, offering ROS developers the opportunity to leverage it for running the ROS framework seamlessly on the Ohmni platform. In addition to the default ohmnilabs/ohmnidev image, an additional docker image, tb\_control, has been introduced to streamline the development process.

This ROS software driver node serves as a crucial component for controlling various actuators on Ohmnilabs robots, including wheels and the neck, while also facilitating sensor feedback acquisition, such as wheel encoder data, neck position, battery voltage, and docking status.

A prerequisite for utilizing this ROS software driver is the possession of an **Ohmni Developer Edition** robot [13]. For a comprehensive understanding of the Ohmni Developer Edition, please refer to the detailed information provided on this webpage: [Develop Robotics application with ROS](#)

### 2.4 Openpose

#### 2.4.1 Introduction

One crucial element in achieving a comprehensive comprehension of individuals depicted in images and videos involves the process of human 2D pose estimation. This process centers on the task of precisely localizing anatomical keypoints or “parts” of the human body. Traditionally, human pose estimation efforts have primarily concentrated on identifying and tracking body parts for single individuals. However, when it comes to inferring the poses of multiple individuals within the same image, a distinct and intricate set of challenges emerges.

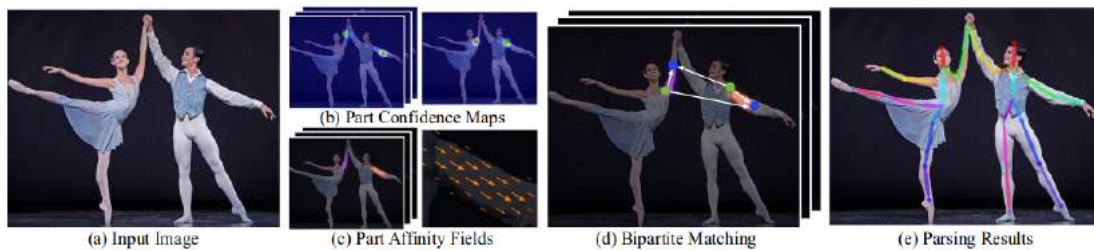
The task of inferring human poses from images presents several significant challenges. Firstly, each image can potentially feature an unpredictable and varying number of individuals, each appearing at different positions and scales. Secondly, the presence of multiple individuals in an image gives rise to intricate spatial interferences, stemming from factors such as physical contact, occlusion, and the articulation of limbs, making it challenging to accurately associate body parts. Lastly, the computational demands during runtime tend to increase with the number of individuals in the image, posing a formidable challenge for achieving real-time performance [9].



Within the confines of this research paper, the Cao et al. [7] introduce an efficient approach to multiperson pose estimation, which demonstrates competitive performance across various publicly available benchmarks. The paper introduces a novel concept: the first-ever bottom-up representation of association scores through Part Affinity Fields (PAFs). These PAFs consist of 2D vector fields that encode both the position and orientation of limbs throughout the image domain.

The paper proceeds to establish that concurrently inferring these bottom-up representations, encompassing detection and association, provides ample global context for a greedy parsing strategy to yield high-quality results. Remarkably, this is accomplished while significantly reducing the computational overhead, thereby offering an approach that is both effective and computationally economical.

### 2.4.2 Method



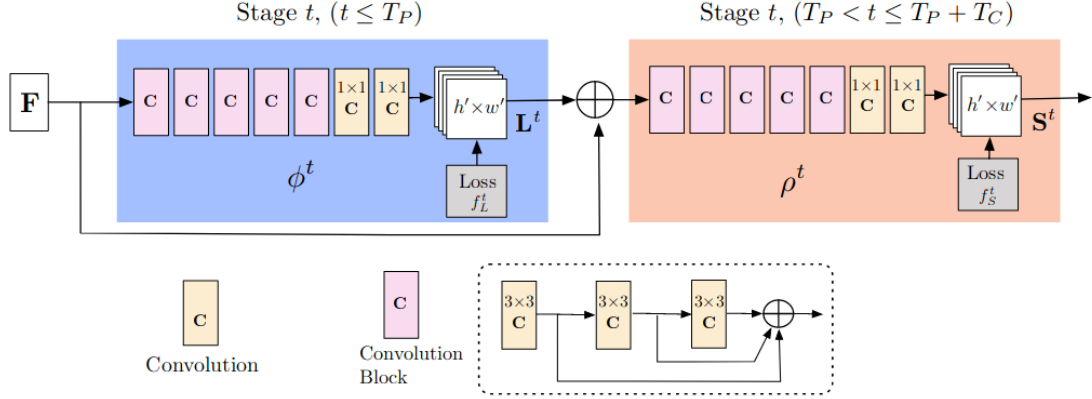
**Figure 2.1:** Overall pipeline.

(a) The method takes the entire image as the input for a CNN to jointly predict (b) confidence maps for body part detection and (c) PAFs for part association. (d) The parsing step performs a set of bipartite matchings to associate body part candidates. (e) Finally assemble them into full body poses for all people in the image [7].

Figure 2.1 illustrates the comprehensive workflow of Openpose method. It commences with the input of a color image with dimensions  $w \times h$  (depicted in Figure 2.1a), ultimately resulting in the determination of 2D anatomical keypoints for each individual within the image (depicted in Figure 2.1e). Initially, a feedforward neural network is employed to make predictions. Specifically, it generates a collection of 2D confidence maps denoted as  $\mathbf{S}$ , which correspond to the spatial locations of body parts (as depicted in Figure 2.1b). Additionally, the network produces a set of 2D vector fields referred to as  $\mathbf{L}$ , representing Part Affinity Fields (PAFs). These PAFs encapsulate information regarding the extent of association between different body parts (as shown in Figure 2.1). Finally, the confidence maps and the PAFs are parsed by greedy inference (Figure 2.1 d) to output the 2D keypoints for all people in the image.

#### 2.4.2.1 Network Architecture

The architectural framework, as illustrated in Figure 2.2, follows an iterative approach for the prediction of affinity fields, denoting part-to-part associations (depicted in blue), and detection confidence maps (displayed in beige). This iterative prediction paradigm progressively enhances



**Figure 2.2:** Architecture of the multi-stage CNN [7].

the quality of predictions across consecutive stages denoted by the parameter  $t \in \{1, \dots, T\}$ . Importantly, intermediate supervision is applied at each of these stages to guide and refine the prediction process.

In the original architectural configuration, the neural network comprised multiple  $7 \times 7$  convolutional layers. In the current model iteration, the receptive field's size is maintained while substantially curtailing computational demands. This reduction in computational load is achieved by substituting each  $7 \times 7$  convolutional kernel with a sequence of three consecutive  $3 \times 3$  kernels. Notably, the computational operations required for the former approach amount to  $2 \times 7^2 - 1 = 97$ , equating to 97 operations. In contrast, the latter method demands a significantly lower computational load of only 51 operations. Furthermore, a novel modification involves the concatenation of the output from each of these three convolutional kernels, drawing inspiration from the principles of DenseNet. This augmentation results in a threefold increase in the number of non-linearity layers, enabling the network to effectively preserve both lower-level and higher-level features.

#### 2.4.2.2 Simultaneous Detection and Association

The input image undergoes an initial analysis through a Convolutional Neural Network (CNN) architecture, which is initialized using the first 10 layers of the VGG-19 model and subsequently fine-tuned for the task at hand. This CNN analysis yields a set of feature maps denoted as  $\mathbf{F}$ , which serves as input to the first stage of our network. In this initial stage, the network generates a set of part affinity fields (PAFs),  $L^1 = \phi^1(F)$

In each subsequent stage of the network, a refined set of predictions is produced. This process leverages the predictions obtained in the previous stage, as well as the original image features  $\mathbf{F}$ . The key to this refinement lies in concatenating these two sources of information and utilizing them to generate more accurate and improved predictions.

$$L^t = (\phi^t)(\mathbf{F}, L^{t-1}), \forall 2 \leq t \leq T_P$$

The symbol  $\phi^t$  denotes the specific Convolutional Neural Networks (CNNs) employed for inference at Stage  $t$ , and  $T_P$  represents the total number of stages dedicated to processing PAFs. This iterative process continues for  $T_P$  iterations. Subsequently, the procedure extends to the detection of confidence maps, commencing with the most up-to-date PAF predictions.

where  $\phi^t$  refers to the CNNs for inference at Stage  $t$ , and  $T_P$  to the number of total PAFs stages. After  $T_P$  iterations, the process is repeated for the confidence maps detection, starting in the most updated PAFs prediction,

$$\mathbf{S}^{T_P} = \rho^t(\mathbf{F}, \mathbf{L}^{T_P}), \forall t = T_P,$$

$$\mathbf{S}^t = \rho^t(\mathbf{F}, \mathbf{L}^{T_P}, \mathbf{S}^{t-1}), \forall T_P < t \leq T_P + T_C,$$

where  $\rho^t$  refers to the CNNs for inference at Stage  $t$ , and  $T_C$  to the number of total confidence map stages.

In order to guide the iterative prediction process, where the network sequentially estimates PAFs for anatomical body parts in the primary branch and confidence maps in the secondary branch, a loss function is systematically applied at the termination of each stage. This loss function is rooted in the  $L_2$  norm, serving as a quantitative measure of the disparity between the network's predictions and the groundtruth representations encoded within the maps and fields. The utilization of a mask plays a pivotal role in training, serving the purpose of preventing the imposition of penalties on genuinely positive predictions. Furthermore, the incorporation of intermediate supervision at each stage serves as an effective strategy for mitigating the vanishing gradient problem, as it periodically replenishes the gradient Cao et al. [7].

#### 2.4.2.3 Confidence Maps for Part Detection

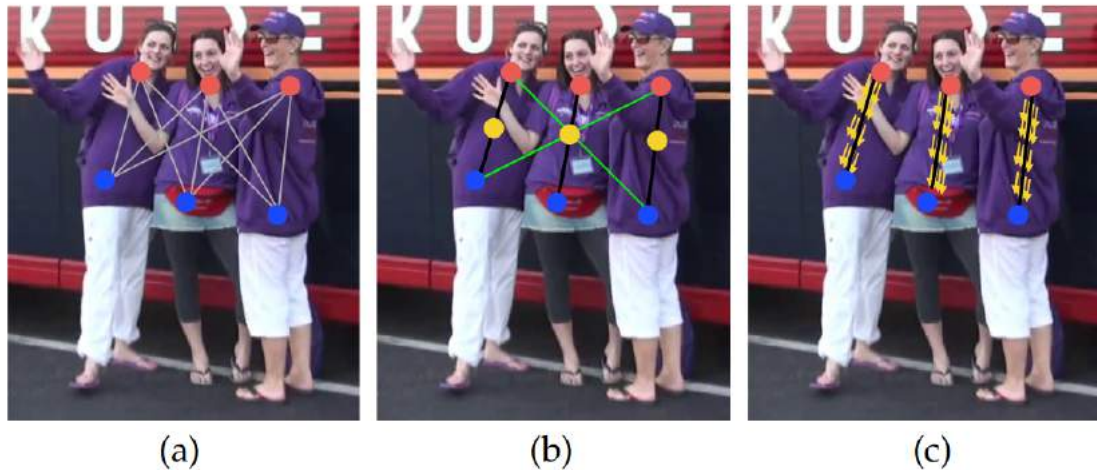
Every confidence map constitutes a 2D representation conveying the degree of certainty regarding the presence of a specific body part at any given pixel location. In an ideal scenario, when a solitary individual is present within the image, each confidence map should exhibit a singular peak for every discernible body part. Conversely, in situations involving multiple individuals within the image, the expectation is for multiple peaks to manifest in each confidence map, each peak corresponding to a visible body part for each distinct person.

The selection of the maximum value from the confidence maps is favored over averaging to ensure that the precision of neighboring peaks retains its distinctiveness, as visually depicted in the right figure. During the testing phase, the prediction of confidence maps is executed, and the identification of potential body part candidates is achieved through the application of a non-maximum suppression technique.

#### 2.4.2.4 Part Affinity Fields for Part Association

In the context of the available set of identified body parts, illustrated as the red and blue data points in Figure 2.3a, the challenge lies in the task of assembling these individual body parts into complete full-body poses for an unspecified number of individuals. A fundamental requirement is the establishment of a confidence metric to assess the association between pairs of detected body

parts, thereby determining whether they pertain to the same individual. One plausible approach to gauge this association involves the detection of intermediary points positioned between each pair of body parts along a limb, as exemplified in Figure 2.3b. However, in densely populated scenarios where individuals congregate closely, these intermediate points are susceptible to yielding erroneous associations, illustrated as the green lines in Figure 2.3b.



**Figure 2.3:** Part association strategies.[7].

These erroneous associations stem from two inherent limitations in the representation:

- It solely encodes the positional information of each limb segment while neglecting their orientation.
- It condenses the region of limb support into a singular point, limiting its effectiveness.

To address these limitations, PAFs are introduced. PAFs offer a solution by preserving both the positional and orientational characteristics of limb segments across their entire support regions, as depicted in Figure 2.3c. In essence, each PAF manifests as a two-dimensional vector field corresponding to a specific limb type, as illustrated in Figure 2.1d. Within the region attributed to a particular limb, these 2D vectors encode the directional information, indicating the orientation from one end of the limb to the other. Consequently, each distinct limb type is associated with a specific PAFs that bridges the two corresponding body parts, enhancing the accuracy of association determination.

### 2.4.3 System

OpenPose addresses several limitations present in existing 2D body pose estimation libraries like Mask R-CNN and Alpha-Pose. These libraries often require users to develop their own components, such as frame readers, result visualization tools, and output file generators. Moreover, they lack integration between facial and body keypoint detection, necessitating the use of multiple separate libraries.

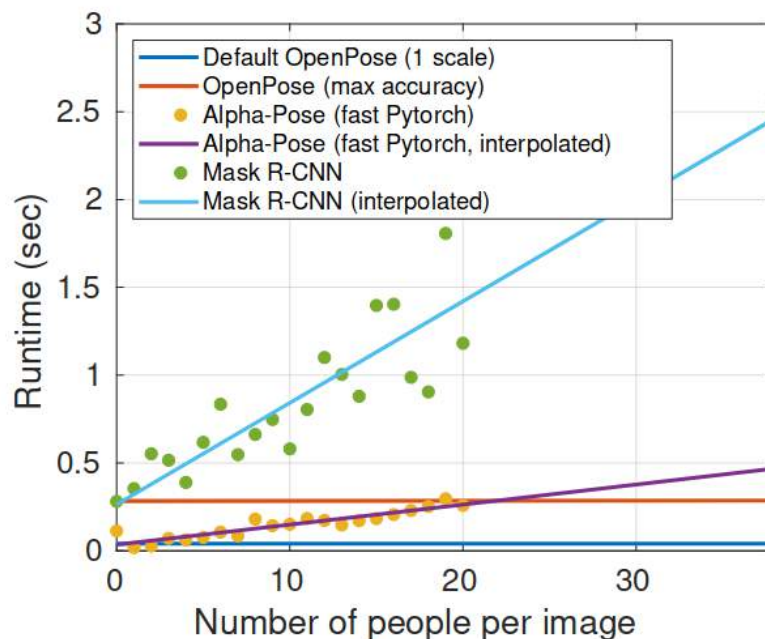
OpenPose offers a comprehensive solution to these challenges. It boasts platform versatility, supporting Ubuntu, Windows, Mac OSX, and even embedded systems like Nvidia Tegra TX2. Additionally, it is compatible with various hardware configurations, including CUDA GPUs, OpenCL GPUs, and CPU-only devices. OpenPose comprises three essential components: body and foot detection, hand detection, and face detection, with the core component being the combined body and foot keypoint detector.

Notably, OpenPose delivers exceptional inference performance, surpassing state-of-the-art methods while maintaining high-quality results. On a Nvidia GTX 1080 Ti-equipped machine, it achieves an impressive frame rate of approximately 22 FPS, all while preserving accuracy. Researchers have already harnessed OpenPose for a wide array of applications, including person re-identification, GAN-based video manipulation of human faces and bodies, Human-Computer Interaction, 3D pose estimation, and 3D human mesh model generation.

## 2.4.4 Evaluation

### 2.4.4.1 Inference Runtime Analysis

This study conducts a comparative analysis among three prominent and widely adopted multi-person pose estimation libraries: OpenPose, Mask R-CNN, and Alpha-Pose. Figure 2.4 illustrates the inference runtime performance of these methods. Notably, the Megvii (Face++) and MSRA GitHub repositories lack information on their person detectors and solely provide pose estimation results for cropped persons, precluding a precise assessment of the runtime performance, rendering their exclusion from this analysis.



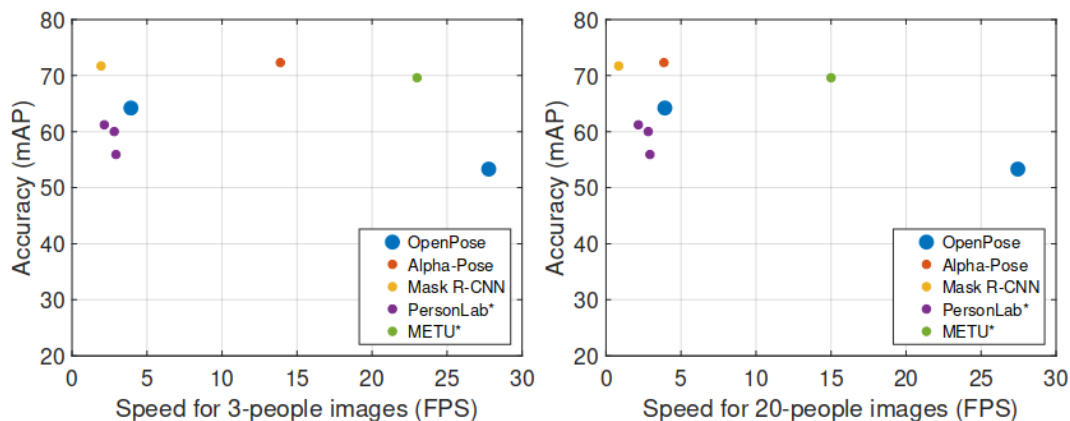
**Figure 2.4:** Inference time comparison between OpenPose, Mask R-CNN, and Alpha-Pose [7].

Mask R-CNN exclusively operates with Nvidia graphics cards, and the analysis is conducted on a system featuring an NVIDIA 1080 Ti. As top-down approaches, the inference times of Mask R-CNN, Alpha-Pose, Megvii, and MSRA are generally proportional to the number of individuals present in the image, specifically corresponding to the number of proposals generated by their respective person detectors. In stark contrast, the bottom-up approach's inference time remains invariant with respect to the number of people in the image.

The runtime of OpenPose comprises two primary components: (1) CNN processing time, characterized by  $O(n^1)$  complexity, remaining constant regardless of the number of people; (2) multi-person parsing time, characterized by  $O(n^2)$  complexity, where 'n' signifies the number of people. However, it's noteworthy that the parsing time is two orders of magnitude shorter than the CNN processing time. For example, parsing requires 0.58 ms for 9 people, whereas CNN processing takes 36 ms.

#### 2.4.4.2 Trade-off between Speed and Accuracy

In the realm of object detection, it is evident that region-proposed methods, exemplified by Faster R-CNN, excel in achieving superior accuracy. Conversely, single-shot methods such as YOLO and SSD exhibit enhanced runtime performance. Similarly, in the context of human pose estimation, a parallel trend emerges, with top-down approaches demonstrating higher accuracy but slower processing speeds when compared to bottom-up methods, particularly in scenarios involving multiple individuals. To offer a comprehensive comparison, this study presents both speed and accuracy metrics for the leading entries in the COCO Challenge, as depicted in Figure 2.5.



**Figure 2.5:** Trade-off between speed and accuracy for the main entries of the COCO Challenge [7].

The principal factor contributing to the diminished accuracy of bottom-up approaches is the inherent limitation in resolution. While top-down methods independently extract and process each detected person within their networks, bottom-up techniques necessitate simultaneous processing of the entire image, resulting in reduced resolution per individual. In summary, top-down methodologies, such as AlphaPose, yield superior results in scenarios with a limited number of individuals, albeit experiencing a notable decrease in processing speed when handling images containing a larger number of people.

Furthermore, it's worth noting that accuracy metrics may potentially be misleading. Nevertheless, the multi-scale approach concurrently delivers both improved speed and accuracy when compared to the versions for which runtime results are reported.

## 2.5 Lidar SLAM

LiDAR SLAM, is a sophisticated procedure employed by robots and autonomous systems to construct a comprehensive map of an unfamiliar environment while concurrently determining their precise position within said map. This technological methodology is underpinned by LiDAR sensors, which emit laser pulses and meticulously measure the time elapsed from pulse emission to its return after encountering objects in the sensor's line of sight. Through an in-depth analysis of the received laser signals, LiDAR SLAM systems adeptly generate intricate 3D point cloud representations that faithfully depict the surrounding environment. Subsequently, this wealth of spatial information is leveraged to precisely gauge the robot's spatial coordinates and orientation relative to the generated map.

The acronym SLAM, which stands for Simultaneous Localization and Mapping, succinctly encapsulates the dual core functions of this process. Localization denotes the pivotal task of pinpointing the precise position and orientation of the robot within the constructed map, while mapping entails the creation of a geometrically accurate representation of the environment. Lidar SLAM systems have gained prominence and garnered considerable attention due to their inherent capacity to deliver exceptionally precise maps in real-time, making them indispensable in a multitude of applications spanning autonomous vehicles, mobile robotics, and indoor mapping, thereby facilitating navigation and asset tracking endeavors [28].

### 2.5.1 Operational principles of Lidar SLAM

- **Principles of Simultaneous Localization and Mapping**

The fundamental premise of SLAM revolves around enabling a robotic or autonomous system to concurrently navigate through an unexplored environment and construct a spatial representation while accurately ascertaining its own positional coordinates within that map. This is achieved by amalgamating data from diverse sensors like Lidar, cameras, and inertial measurement units (IMUs), collectively aiding in trajectory estimation and landmark localization.

SLAM algorithms typically comprise two essential components: the prediction step, responsible for updating the robot's position based on past position and sensor-derived motion data, and the correction step, which refines the predicted position using measurements from environmental cues, such as distances to landmarks sensed by Lidar.

Through iterative execution of prediction and correction steps, SLAM algorithms iteratively refine the robot's position and the environment map. The resulting map can take various forms, including occupancy grids, point clouds, or graphs, encapsulating the spatial relationships among detected landmarks.

- **Role of Lidar Sensors in SLAM**

Lidar sensors assume a pivotal role within SLAM systems by furnishing precise and high-resolution environmental data. These sensors emit laser pulses, which subsequently rebound off objects and return to the sensor, facilitating distance measurements based on pulse travel time. Lidar sensors excel in generating dense point clouds, comprising 3D spatial points that delineate object surfaces within the environment.

The point cloud data gleaned from Lidar sensors serve as valuable resources for identifying and tracking landmarks, such as walls, corners, or other distinctive environmental features. These landmarks hold paramount significance for SLAM algorithms, as they furnish vital information to refine the robot's estimated position and orientation. Additionally, Lidar data can be harnessed for obstacle detection and avoidance, rendering it especially invaluable for autonomous navigation and ensuring safe operation.

- **Map Generation and Pose Estimation**

LiDAR SLAM is fundamentally concerned with two core tasks: the creation of detailed maps using Lidar data and the concurrent estimation of the robot's position within these maps. Maps can be represented in various formats, such as occupancy grids or point clouds.

To estimate the robot's position, SLAM algorithms employ Lidar data to identify and track environmental landmarks. By comparing observed landmarks to those stored in the map, the algorithm deduces the robot's position and orientation relative to the map. This involves finding the optimal transformation that aligns the observed Lidar data with the map, often achieved through techniques like iterative closest point (ICP) or optimization algorithms.

As the robot explores its surroundings and collects new Lidar data, the map undergoes continuous updates, thereby refining the robot's pose. This iterative process empowers Lidar SLAM systems to generate accurate maps and provide real-time localization, serving various applications, including autonomous vehicles, mobile robotics, and indoor mapping [28].

### 2.5.2 Hector SLAM

Hector SLAM is a Lidar-based Simultaneous Localization and Mapping (SLAM) algorithm notable for its independence from odometry data, rendering it suitable for platforms lacking wheel encoders or motion sensors. This algorithm relies on a grid map representation and employs a multi-resolution approach to process Lidar data at varying resolutions, facilitating efficient navigation in expansive environments [19].

At the heart of Hector SLAM lies a rapid scan matching technique, aligning consecutive Lidar scans to estimate the robot's motion accurately. To mitigate accumulated pose estimation errors over time, the algorithm incorporates loop closure detection and optimization mechanisms. Hector SLAM has demonstrated successful applications across diverse platforms, encompassing aerial vehicles, ground-based robots, and even handheld Lidar scanners [28] [2].



## 2.6 Localization - AMCL

- **Particle Filter**

In the context of particle filters, a substantial quantity of particles spanning the entire state-space is employed to establish the filtering process. With an accumulation of additional data, predictions and measurement updates are iteratively performed, resulting in the emergence of a multi-modal posterior distribution for the robot's state. This approach stands in stark contrast to the Kalman Filter, which approximates the posterior distribution as Gaussian and converges towards a single state-space value after several iterations.

The particle filter methodology involves the following key steps:

1. **Re-sampling:** This step entails drawing random samples from the existing set of particles, allowing replacement. The selection of particles is influenced by their associated importance weights. Particles with higher weights are favored and drawn more frequently, while those with lower weights may be drawn minimally or not at all. Subsequently, all particles are assigned equal weightage post-re-sampling.
2. **Sampling:** To sample from the distribution characterizing the system's dynamics, past beliefs and control information are taken into account. The current belief is represented by the density obtained through the product of the distribution and a prior belief instance. This density serves as the basis for the proposal distribution in the subsequent stage.
3. **Importance Sampling:** In this phase, each sample is weighted by its importance weight, which corresponds to the likelihood of the sample given the measurement. This step effectively incorporates the measurement information into the belief update process.

Through these three sequential procedures, samples are generated to represent the posterior belief in each iteration. Furthermore, the relevance weights of the data are periodically adjusted after 'n' repetitions to ensure that they collectively sum to 1 [AMCL] [47].

- **Adaptive Particle Filter for Robot Localization**

Particle filters struggle with random particle distribution, especially in high-dimensional problems. Adaptive particle filters outperform standard ones by approximating the true posterior with a piecewise constant distribution, reducing inaccuracies. The number of particles needed is inversely related to a specified accuracy threshold.

For localization, adaptive particle filters begin with an environment map. The robot is placed manually or starts with no initial estimate. It predicts its position with new samples after each motion, integrates sensor readings, and benefits from randomly added samples for recovery. Symmetry-induced map ambiguities lead to a multi-modal posterior, requiring numerous sensor readings for convergence [2].



## 3 State of the Art

Our objective is to craft a software architecture comprising interconnected modules for enhancing the navigation capabilities of our Ohmni telepresence robot within indoor environments, enabling seamless execution of complex tasks. The focal point of the Master's Thesis revolves around the development and integration of diverse modules encompassing image analysis, decision-making, human-robot interaction, and autonomous navigation. These individual modules necessitate precise orchestration to form a cohesive and comprehensive architecture. With this objective in mind, we examined prior implementations across different robotic platforms. The goal was to assimilate these concepts and tailor them to suit the unique characteristics of the Ohmni telepresence robot.

### 3.1 Environmental Awareness in Mobile Service Robots

Comprehending the environment stands as a pivotal capability for mobile service robots in their journey towards autonomy. Despite its inherent difficulty, researchers have devised an array of solutions aimed at empowering robots to engage with and gain insights into their surrounding environment. Prasad and Ertel [42] provides an in-depth review of contemporary artificial intelligence methodologies employed by researchers. These methodologies are focused on facilitating robots in the acquisition and processing of knowledge pertaining to their environment and contextual circumstances, thereby fostering a sense of awareness in robotics. Jumel et al. [22] delves into a comprehensive discussion regarding the fundamental capabilities expected from a humanoid service robot. This discussion revolves around the utilization of contemporary methodologies associated with social navigation and deep learning to construct these capabilities. The paper presents a meticulous architectural framework that elucidates the interconnection between high-level functionalities and their underlying low-level sub-functions Robinson et al. [44].

The article Moniz and Krings [31], undertakes an examination of two hypotheses. The first hypothesis centers on the identification of pertinent research inquiries that pertain to the potential integration of robotic systems within various modes of work organization on the manufacturing shop-floor level. The second hypothesis involves a comprehensive discussion regarding the conceptualization of past organizational challenges related to HRI. Within this framework, the article contemplates the limitations posed by cognitive and perceptual workloads for robot operators functioning within complex operational systems.

### 3.2 Elevating Human-Robot Interaction

Enabling machines to develop an understanding of individuals in images and videos hinges on the real-time estimation of multi-person 2D poses. This study Cao et al. [8] presents an approach designed for real-time detection of the 2D poses of multiple individuals within images. The method

in question utilizes a nonparametric representation referred to as PAFs to establish associations between body parts and individuals depicted in the images. Operating on a bottom-up methodology, this system achieves both high accuracy and real-time performance, regardless of the number of individuals present in the images.

The integration of the OpenPose software framework with the ROS signifies a significant advancement in the field of human-robot interaction and perception. This amalgamation effectively merges the advanced human pose estimation capabilities of OpenPose with the extensive robotic functionalities provided by ROS. Through the incorporation of ROS and OpenPose, the robot's perceptual abilities are greatly enhanced, opening the door to a wide range of applications, including but not limited to, human-robot collaboration, assistive robotics, interactive entertainment, and healthcare support. Joshi et al. [21] thoroughly elucidates the intricacies involved in integrating OpenPose with ROS, emphasizing the seamless interoperability achieved, and explores the consequential implications for elevating the overall quality of the human-robot interaction experience.

The approach presented by Martinelli et al. [29], in this context offers a solution by utilizing computer vision techniques to create a gesture control interface. While the field of mobile robotics predominantly advances through autonomous systems, there are situations where this strategy is impractical [27], [43]. In these instances, non-autonomous robots must be operated using human-machine interfaces, often relying on costly industrial equipment. The utilization of fixed control mechanisms to govern robot actions restricts operators to the machinery [62], resulting in reduced mobility and agility. Furthermore, these methods often lead to a lack of control precision. De Assis Moura Pimentel and Aquino [12] addresses the ongoing challenge of enabling social navigation in service robots. It provides a comprehensive review of the latest social navigation models while introducing an innovative approach—a novel social navigation model that integrates context extraction from ontologies. This advancement is anticipated to enhance not only the robot's authenticity and sociability but also to improve the overall comfort of human-robot interactions.

### **3.3 Human-Robot Synergy for Enhanced Exploration**

Xia et al. [65] outlines a mobile robot system meticulously designed to collaboratively explore previously uncharted environments in the company of humans. This feat is achieved through a seamless integration of simultaneous localization and mapping SLAM techniques, strategic motion planning, and accurate human recognition capabilities. Moreover, the system ensures that the robot exhibits a heightened awareness of human presence, thereby maintaining a behavior that is attuned to human safety, individual preferences, and interactive dynamics throughout the exploration process [58]. The system demonstrates the ability to simultaneously accomplish essential tasks such as robot localization, mapping, and human tracking [5].

Yuan et al. [72] aims to present a viable approach for the integration of these three facets, culminating in a navigation system that embodies consciousness, safety, accuracy, robustness, and efficiency. The navigation challenges pertaining to mobile service robots are distilled into three key components: 1. Human detection, 2. Real-time robot localization, and 3. Robot motion planning. Furthermore, Miller et al. [30] provides an all-encompassing overview of an autonomy solution designed for mobile robots. It expounds upon an approach aimed at enhancing human-following operations and delves into an investigation of a strategy for automated decision-making in parking scenarios.

The thorough examination of existing literature has equipped the author with crucial insights to initiate their work, while also facilitating a comprehensive understanding of how research in this field has evolved from its foundational principles to the present. Moreover, these literature surveys have played a pivotal role in shaping the research methodology and strengthening the concepts intended for implementation in the Master's Thesis. The unresolved issues highlighted as limitations or drawbacks in prior research endeavors now serve as potential focal points for the Master's Thesis. A profound grasp of the preceding work in this domain was indispensable, as progressing with the research would have been unfeasible without it.



## 4 Design of Solution

### 4.1 Hardware Setup

The hardware configuration is elucidated comprehensively through the illustrative representation provided in the Figure 1.1.

#### 4.1.1 Microsoft Kinect RGB-D Camera v1

- **Camera(RGB) Specifications:**The Kinect camera is classified as an RGB (Red-Green-Blue) device with a resolution of 640×480 pixels and a color range encompassing 24 bits. Operating at a frame capture rate of 30 frames per second (fps), this camera is analogous to conventional webcams or the sensors commonly found in digital cameras, rendering it quite ordinary in most aspects.

While the camera possesses the capability to capture imagery at a higher resolution of 1280×960 pixels, it becomes apparent that the maximum attainable frame rate at this elevated definition is approximately 15fps. Conversely, faster frame rates can be achieved by reducing the camera's resolution.

The operational scope of the Kinect sensor is confined to a specified distance range, spanning from 1.2 to 3.5 meters. Its horizontal field of view spans 57 degrees, signifying that at its maximum range, it can scan a scene with a width of 3.8 meters.

The sensor boasts a vertical field of view measuring 43 degrees, equivalent to 63 centimeters (25 inches) in physical dimensions. This field of view can be further augmented by utilizing the vertical pivot system, which permits the sensor to tilt up or down by a maximum of 27 degrees in either direction [60].



**Figure 4.1:** Kinect RGB Camera [60]



**Figure 4.2:** Kinect IR Camera [60]



**Figure 4.3:** Kinect IR Projector [60]

- **IR Depth Sensor Specifications:** The Kinect's depth measurement capabilities are attributed to the presence of an infrared (IR) emitter and an IR depth sensor. The emitter projects infrared light beams, and the depth sensor subsequently records the reflected IR beams. These reflected beams are then transformed into depth information, effectively quantifying the spatial gap between an object and the sensor. This pivotal process enables the acquisition of depth images.

The depth sensor system is comprised of an infrared projector and a monochrome CMOS sensor, positioned on either side of the RGB camera. Notably, the depth sensor exhibits a resolution of 640×480 pixels and is endowed with 16-bit sensitivity. In practical terms, this equates to an ability to discern approximately 65,000 distinct shades of gray.

Similar to the RGB camera, the depth sensor is also capable of capturing video at a consistent rate of 30 frames per second. However, it is worth noting that frame drops may occur contingent upon the specific hardware and computational demands associated with the task being performed [60].

Some rough estimates of the accuracy of the depth sensor:

**Range:** 50 cm to 5 m. Can get closer (40 cm) in parts, but can't have the full view be < 50 cm.

**Horizontal Resolution:** 640 x 480 and 45 degrees vertical Field of View (FOV) and 58 degrees horizontal FOV. Simple geometry shows this equates to about 0.75 mm per pixel x by y at 50 cm, and 3 mm per pixel x by y at 2 m.

**Depth resolution:** 1.5 mm at 50 cm. About 5 cm at 5 m.

**Noise:** About ±1 Depth Noise (DN) at all depths, but DN to depth is non-linear. This means ±1 mm close, and ±5 cm far [60].

- **Camera Portability Modification:** In order to adapt the Kinect Camera for our specific use case, it necessitates hardware modifications to enhance its portability and versatility. A comprehensive explanation of these modifications can be found in a YouTube video presented by Mr. Nasiru Aboki [Kinect XBOX 360 Custom Mobile Adapter](#) [24].

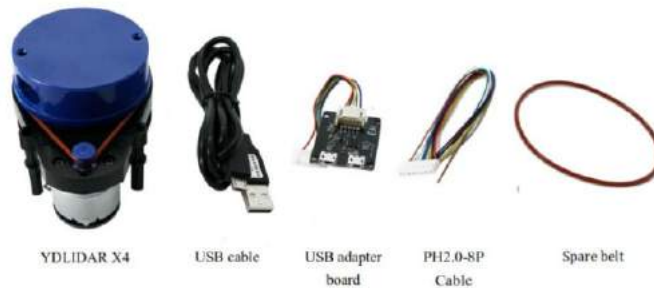
### 4.1.2 YD LiDAR X4

The YDLIDAR X4 represents an affordable 2D LiDAR solution boasting a 360-degree scanning range, retailing at a mere \$99. In the realm of cost-effective LiDARs, only a select few, such as the RPLidar A1M8, can rival its economic appeal. A compelling reason to opt for the YDLIDAR X4 lies in its seamless compatibility with the Robot Operating System (ROS) and the Ohmni robot platform.

To establish a functional connection, one should initiate the assembly process by interconnecting the adapter board with the YDLIDAR X4. Subsequently, the Universal Serial Bus (USB) cord should be affixed to the USB port on the adapter board, which is then connected to a Personal Computer (PC). Notably, the USB interface's Micro interface must be linked to the USB DATA port on the USB adapter [68].

Upon activation, the YDLIDAR X4 enters an idle state, wherein its motor remains stationary. However, it is imperative to acknowledge that some development platforms or PC USB interfaces may not provide adequate drive current for the YDLIDAR X4's optimal operation. To rectify this, it





**Figure 4.4:** YDLIDAR X4 Development Kit

becomes necessary to supply the LiDAR with an auxiliary +5V power source. In such scenarios, the USB-PWR pin emerges as a viable means to deliver the requisite power supply [X4 User Manual \[70\]](#).

It is worth noting that when connecting the YDLIDAR X4 to the local host system or laptop, only the utilization of the USB DATA connection point is necessary to establish a functional link [X4 Development Manual \[69\]](#).

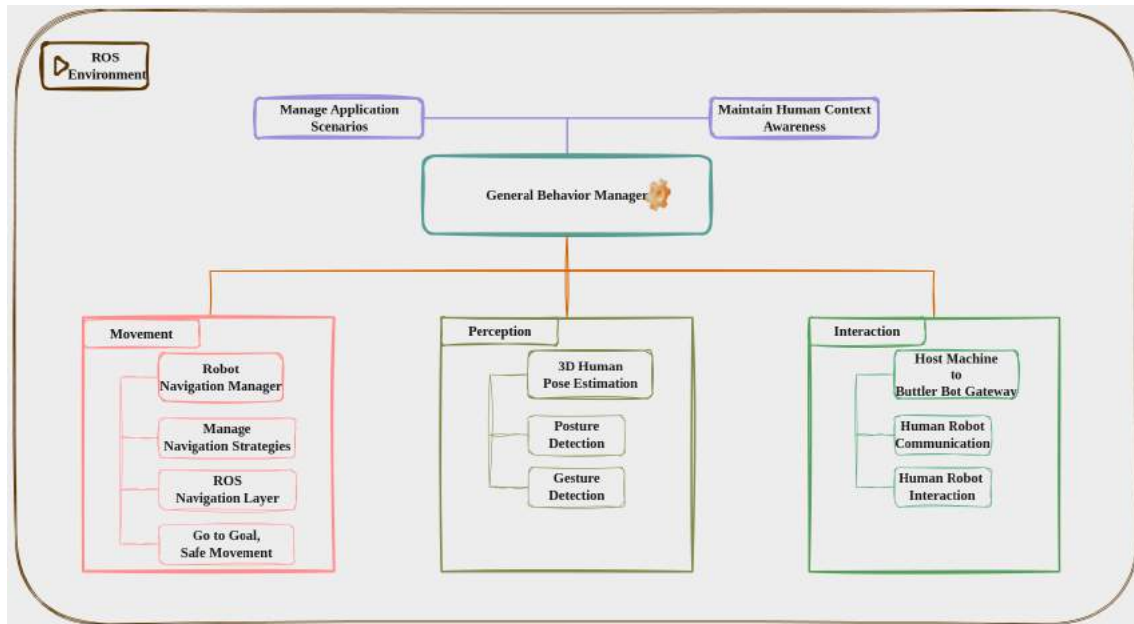
### 4.1.3 System Hardware Specifications

The local host machine under consideration for this research exhibits a noteworthy hardware configuration. It boasts a total system memory (RAM) capacity of 7.2 gibibytes (GiB), providing ample resources for data-intensive computational tasks. Powering the system is the AMD® Ryzen 5 4600H processor, a formidable choice featuring 12 processing cores that contribute to enhanced computational efficiency. In terms of storage capacity, the local host machine offers an expansive 512.1 gibibytes (GB) of disk space, ensuring ample room for data storage and retrieval. Graphics processing capabilities are reinforced by the presence of the NVIDIA GeForce GTX 1650 graphics processor, which offers substantial support for graphical tasks. Notably, the local host machine allocates 3911 megabytes (MB) of dedicated memory for graphics processing tasks, further enhancing its ability to handle graphical workloads. This hardware configuration is well-suited for a range of scientific and computational applications, making it an ideal choice for research endeavors that demand substantial computational power and storage capacity.

## 4.2 Towards seamless HRI

The General Behavior Manager in Figure 4.5 functions as an orchestrator, facilitating communication among various modules to accomplish high-level tasks. These modules collaborate to achieve complex objectives, as elaborated below.

The Movement Module in Figure 4.5 assumes responsibility for robot localization and dynamic navigation, including obstacle avoidance. It leverages two sub-functions, namely, the ROS Navigation layer and the ROS Mapping layer, to execute these tasks effectively.



**Figure 4.5:** Software Architecture Overview

The Perception Module in Figure 4.5 specializes in 3D Human pose estimation, employing Openpose technology. This capability enables the estimation of human postures and gestures with precision.

All data processing occurs on our local host machine. The Interaction Module in Figure 4.5 plays a pivotal role in guiding the robot’s decision-making process in diverse scenarios and conditions, all while maintaining an acute awareness of its surroundings.

It is noteworthy that these modules operate seamlessly within the ROS (Robot Operating System) environment.

#### 4.2.1 Perception

The pivotal perceptual elements within the software stack encompass pose estimation, SLAM, and localization AMCL.

- **Openpose**

The process of pose estimation and human skeletal frame detection is executed through the utilization of the Openpose library, which serves as a critical component within the software framework. This intricate procedure involves the extraction of both 2D and 3D coordinates corresponding to key human joint pixels, directly derived from images. These extracted coordinates provide a comprehensive representation of the human posture and skeletal structure.

By harnessing this rich data, the system achieves a remarkable level of sophistication in recognizing various human poses. These poses encompass a wide spectrum, ranging from fundamental states such as sitting and standing, to more nuanced orientations in relation to the robot’s position, which includes facing towards, sideways, and even backward.

Moreover, the system extends its capabilities beyond mere pose recognition. It leverages hand gestures as a means of controlling the robot's movements. These gestures enable the robot to execute actions such as moving forward, reversing, and performing turns. This integration of gesture control adds a layer of intuitive and interactive functionality to the system.

A notable application of this technology is its integration with a navigation stack as discussed in section 4.2.1 SLAM and Localization. In this context, human posture and gesture information harmoniously interface with the robot's navigation system. This synergy empowers the robot to autonomously navigate within a mapped environment, responding to high-level commands and user intentions. Consequently, the robot can seamlessly move towards predefined destinations, offering a sophisticated and adaptable solution for real-world scenarios where human-robot interaction is essential.

- **SLAM and Localization**

Within the software stack, the intricate process of Simultaneous Localization and Mapping (SLAM) is realized through the utilization of the Hector SLAM algorithm, a robust technique that produces a 2D occupancy grid map. This map serves as a foundational representation of the environment, enabling the robot to discern its position and surroundings with high precision. Following the generation of this map, localization tasks are seamlessly performed using the Adaptive Monte Carlo Localization (AMCL) algorithm. [AMCL](#) [47] integrates sensor data and effectively localizes the robot within the 2D occupancy grid, enhancing the robot's situational awareness and navigation capabilities.

The software stack's planning functionalities encompass two fundamental components: obstacle detection and global path planning. Obstacle detection leverages data acquired from the 2D Lidar sensor, allowing for the creation of a 2D occupancy grid costmap through the [costmap\\_2d](#) [51] ROS package. This costmap meticulously captures information about obstacles in the robot's vicinity, facilitating obstacle avoidance and safe navigation.

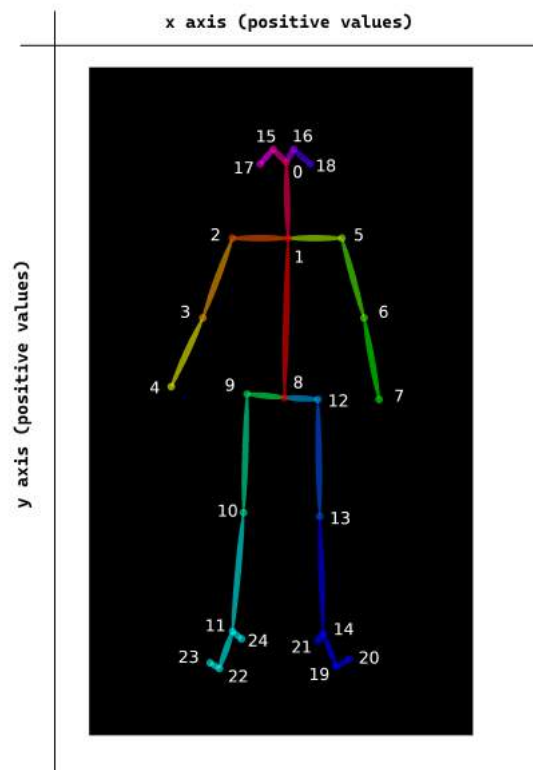
Global path planning, a critical aspect of autonomous navigation, is orchestrated by the [global\\_planner](#) [53] ROS package. This planning module undertakes the intricate task of charting a global trajectory for the robot. It calculates a path from the robot's current position to its designated goal while judiciously considering the presence of static obstacles within the environment. This ensures that the robot can plan and execute safe and efficient routes to its destinations.

In the domain of control, the software stack incorporates two pivotal components: a local path planner and a low-level controller, both skillfully executed through the [dwa\\_local\\_planner](#) [52] ROS package. The local path planner plays a pivotal role in generating dynamic trajectories that align with the global trajectory. Importantly, it adapts to the presence of dynamic obstacles, allowing the robot to make real-time adjustments to its path to avoid collisions.

Concurrently, the low-level controller employs a Dynamic Window Approach to sample and execute collision-free velocity commands. This approach leverages sensor data and the current state of the robot to make rapid and precise control decisions. It ensures that the robot's movements are not only responsive but also safe, enabling it to navigate through complex and dynamic environments with agility and reliability.

### 4.3 Gesture and Posture Detection

The research presents a novel approach to HRI based on human gestures and postures for task assignment to the robot. The primary emphasis lies in the seamless integration of gestures and postures into the issuance of task commands.



**Figure 4.6:** Output from OpenPose.

The numbered coordinates correspond to the following joint positions: 0 - Nose, 1 - Neck, 2 - Right Shoulder, 3 - Right Elbow, 4 - Right Wrist, 5 - Left Shoulder, 6 - Left Elbow, 7 - Left Wrist, 8 - MidHip, 9 - Right Hip, 10 - Right Knee, 11 - Right Ankle, 12 - Left Hip, 13 - Left Knee, 14 - Left Ankle, 15 - Right Eye, 16 - Left Eye, 17 - Right Ear, 18 - Left Ear, 19 - Left Big Toe, 20 - Left Small Toe, 21 - Left Heel, 22 - Right Big Toe, 23 - Right Small Toe, 24 - Right Heel, 25 - Background

#### 4.3.1 Gesture Estimation from Pose Implementation

In our research, **static gestures** are determined through camera-based pose estimation. When provided with an image of a human subject, the OpenPose library [21] furnishes estimates for the pixel coordinates of each joint in both 2D and 3D dimensions within the image. The Figure 4.6 presented illustrates the labels assigned to these joint coordinates as generated by OpenPose. Each joint label is associated with estimated pixel coordinates denoted as 'x' and 'y,' with the origin

set at the upper left pixel, as depicted in the Figure 4.6. It is crucial to note that both 'x' and 'y' coordinates exclusively hold **positive values**. Subsequently, gestures are inferred by analyzing the geometric relationships among specific body joints.

We have established a set of eight distinct gestures represented as **Boolean values**, wherein each specific gesture is assigned a value of **1** when actively executed and **0** when not in use. It is important to note that this research primarily concentrates on arm gestures, despite the existence of numerous potential static gestures. The following gestures, each with its respective function, have been implemented on the Butler Robot:

- **obey\_gesture**: This gesture serves as a signal to instruct the robot to respond to the user's commands conveyed through gestures.
- **disobey\_gesture**: This particular gesture conveys to the robot the instruction to disregard the commands issued by the user through gestures.
- **go\_forward\_gesture**: Directs the robot to move forward.
- **go\_backward\_gesture**: Commands the robot to move in reverse.
- **go\_right\_gesture**: Instructs the robot to make a right turn.
- **go\_left\_gesture**: Directs the robot to make a left turn.
- **go\_forward\_with\_right\_turn**: Combines forward movement with a right turn.
- **go\_forward\_with\_left\_turn**: Combines forward movement with a left turn.

The smooth functioning of controlling the robot with gestures is facilitated by a set of defined mathematical equations corresponding to each of the aforementioned gestures.

For the sake of simplicity and precision in scientific communication, it is essential to establish a set of variables with clear and distinct representations. Herein, we introduce the following symbolic definitions:

- "L" shall be utilized to represent the left side of the Human.
- "R" shall be designated for the right side of the Human.
- "W" shall denote the wrist of the Human under consideration.
- "S" shall be indicative of the shoulder.
- "E" shall specifically refer to the elbow.
- "H" shall signify the hip.
- "A" shall be employed to represent the ankle.
- "K" shall be reserved for the knee joint.
- "x" shall serve as the variable denoting the horizontal x-coordinate.
- "y" shall be used to signify the vertical y-coordinate.

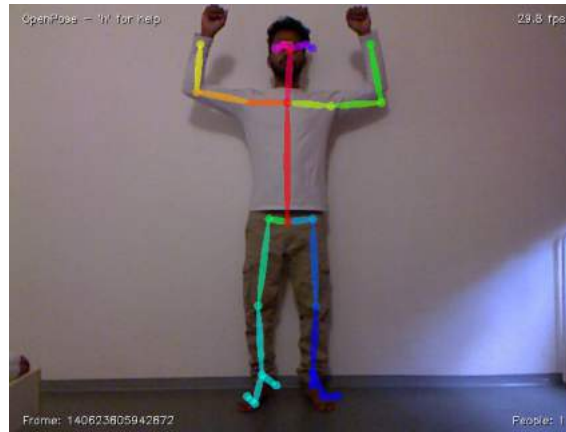
By adhering to these carefully defined symbols, we can enhance the precision and clarity of scientific discourse, promoting effective communication within the scientific community.

To illustrate this concept further, consider the following instances:  $LS_x$  signifies the x-coordinate corresponding to the left shoulder of a detected human, while  $RH_y$  designates the y-coordinate pertaining to the right hip of a detected individual.

Images depicting various poses accompany each mathematical equation, enhancing the comprehension of the mathematical expressions through visual gestures.

$$(4.1) \text{ obey\_gesture} = ((LS_y > LW_y) \vee (RS_y > RW_y))$$

,where the Figure 4.7 serves as a representation of the Obey gesture.



**Figure 4.7:** Obey Gesture

In order to enhance the readability of the lengthy Gesture equations, we will break them down into smaller components to facilitate comprehension.

$$(4.2) \text{ disobey\_gesture} = (\text{Component\_A}) \wedge (\text{Component\_B}) \wedge (\text{Component\_C}) \wedge (\text{Component\_D})$$

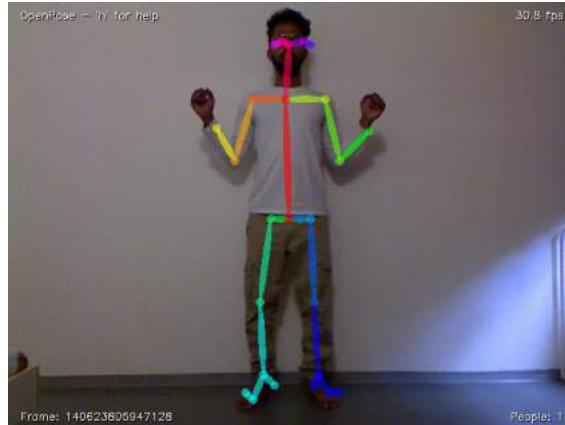
In the provided equation (4.2), the components are delineated as follows, where the Figure 4.8 serves as a representation of the Disobey gesture.

$$\text{Component\_A} = (RE_y > RS_y > 0) \wedge (RE_y > RW_y > 0)$$

$$\text{Component\_B} = (|RS_x - RW_x| < 50) \wedge (|RE_y - RW_y| < 50)$$

$$\text{Component\_C} = (LE_y > LS_y > 0) \wedge (LE_y > LW_y > 0)$$

$$\text{Component\_D} = (LW_x - RW_x) > (LS_y - RS_x) \wedge (LW_x > Neck_x > RW_x)$$



**Figure 4.8:** Disobey Gesture

$$(4.3) \text{ go\_forward} = (\text{Component\_A}) \wedge (\text{Component\_B}) \wedge (\text{Component\_C}) \wedge (\text{Component\_D})$$

representing the “go\_forward\_gesture” as simply “go\_forward”

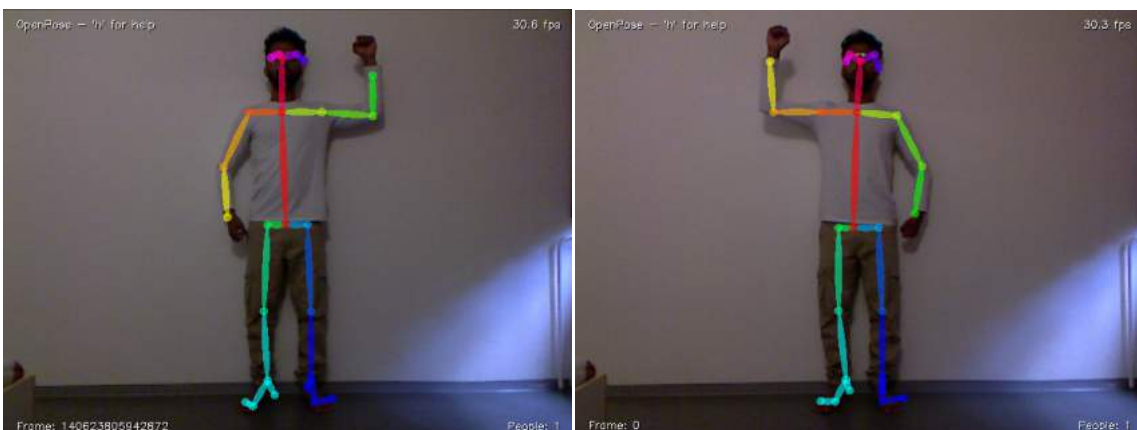
In the provided equation (4.3), the components are delineated as follows, where the Figure’s 4.9 and 4.10 serves as a representation of the Go forward gesture.

$$\text{Component\_A} = ((LS_y > LW_y > 0) \vee (RS_y > RW_y > 0)) \wedge \neg((LS_y > LW_y > 0) \wedge (RS_y > RW_y > 0))$$

$$\text{Component\_B} = (RS_x > RW_x > 0) \vee (LW_x > LS_x > 0)$$

$$\text{Component\_C} = (|RW_x - RE_x| < 100) \vee (|LW_x - LE_x| < 100)$$

$$\text{Component\_D} = (|RW_y - RH_y| < 100) \vee (|LW_y - LH_y| < 100)$$



**Figure 4.9:** Go forward gesture with left hand **Figure 4.10:** Go forward gesture with right hand

$$(4.4) \text{ go\_backward} = (\text{Component\_A}) \wedge (\text{Component\_B}) \wedge (\text{Component\_C}) \wedge (\text{Component\_D})$$

representing the “go\_backward\_gesture” as simply “go\_backward”

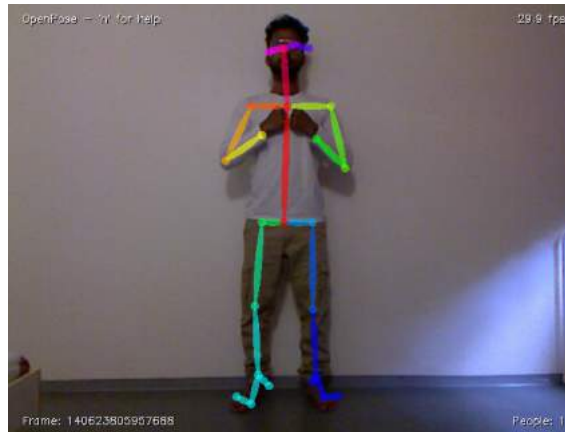
In the provided equation (4.4), the components are delineated as follows, where the Figure 4.11 serves as a representation of the Go backward gesture.

$$\text{Component\_A} = ((RE_y > RS_y > 0) \wedge (RE_y > RW_y > 0))$$

$$\text{Component\_B} = ((LW_x > Neck_x > RW_x) \wedge (LE_y > LS_y > 0))$$

$$\text{Component\_C} = (LE_y > LW_y > 0)$$

$$\text{Component\_D} = (LW_x - RW_x) < (LS_x - RS_x)$$



**Figure 4.11:** Go backward gesture

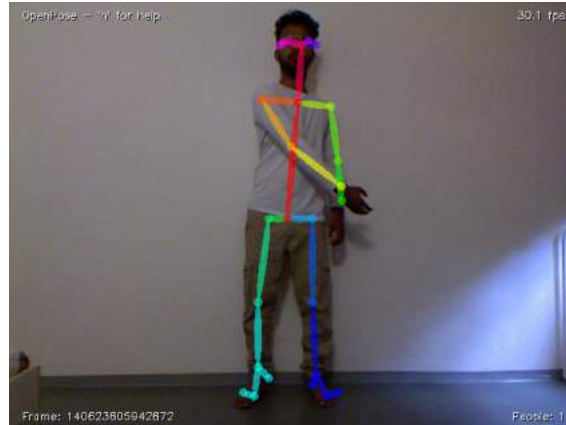
$$(4.5) \text{ go\_right\_gesture} = (\text{Component\_A}) \wedge (\text{Component\_B})$$

In the provided equation (4.5), the components are delineated as follows, where the Figure 4.12 serves as a representation of the Go right gesture.

$$\text{Component\_A} = (0 < Neck_x < RW_x) \wedge (0 < LS_x < LE_x < LW_x)$$

$$\text{Component\_B} = (0 < LS_y < LW_y) \wedge (0 < RS_y < RW_y)$$





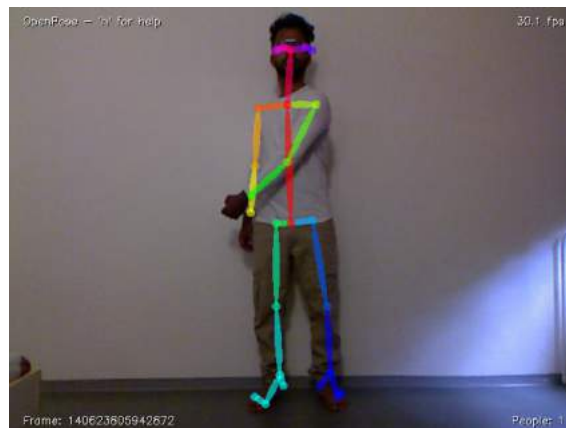
**Figure 4.12:** Go right gesture

$$(4.6) \text{ go\_left\_gesture} = (\text{Component\_A}) \wedge (\text{Component\_B})$$

In the provided equation (4.6), the components are delineated as follows, where the Figure 4.13 serves as a representation of the Go left gesture.

$$\text{Component\_A} = (0 < LW_x < Neck_x) \wedge (0 < RW_x < RE_x < RS_x)$$

$$\text{Component\_B} = (0 < LS_y < LW_y) \wedge (0 < RS_y < RW_y)$$



**Figure 4.13:** Go left gesture

$$(4.7) \text{ forward\_right} = (0 < RS_y < Neck_y < LS_y) \wedge (0 < RH_x < RS_x) \wedge (0 < LH_x < Neck_x)$$

representing the “go\_forward\_with\_right\_turn” as simply “forward\_right”, where the Figure 4.14 serves as a representation of the Go forward with right turn gesture.



**Figure 4.14:** Go forward with right turn gesture

$$(4.8) \text{ forward\_left} = (0 < LS_y < Neck_y < RS_y) \wedge (0 < LS_x < LH_x) \wedge (0 < Neck_x < RH_x)$$

representing the “go\_forward\_with\_left\_turn” as simply “forward\_left”, where the Figure 4.15 serves as a representation of the Go forward with left turn gesture.



**Figure 4.15:** Go forward with left turn gesture

To govern the robot’s motion, we employ the publication of linear and angular velocity commands via the `/tb_cmd_vel` ROS topic. As an illustrative scenario with python script, consider the task of directing the robot to move forward while simultaneously executing a left turn.

```
def forward_left_turn(self):
    twist = Twist()
    twist.linear.x = max_linear_vel
    twist.linear.y = 0
    twist.angular.z = max_angular_vel
    self.pub.publish(twist)
```

In this script, the code initializes a ROS publisher named 'self.pub' with the topic '/tb\_cmd\_vel' and a message type of 'Twist'. Publishers are used to send messages to specific topics in the ROS ecosystem. The 'queue\_size' parameter specifies the maximum number of messages that can be stored in the publisher's outgoing message queue. A 'Twist' message is then created and configured with values for linear and angular velocity. The 'twist.linear.x' attribute is set to 'max\_linear\_vel = 0.2' and 'twist.angular.z' is set to 'max\_angular\_vel = 0.2'. These values likely represent the desired linear and angular velocities for a robot. Finally, the 'publish' method is called on the 'self.pub' publisher object to send the 'twist' message to the '/tb\_cmd\_vel' topic, effectively instructing a robot or another ROS node to move with the specified linear and angular velocities.

### 4.3.2 Posture Estimation from Pose Implementation

In a manner similar to how static gestures are estimated, we utilize camera-based pose estimation to discern various human postures. When presented with an image of a human subject, the OpenPose library [21] provides estimations for the pixel coordinates of each joint in both 2D and 3D dimensions within the image. The accompanying Figure 4.6 illustrates the labels assigned to these joint coordinates as generated by OpenPose. Each joint label is associated with estimated pixel coordinates represented as 'x' and 'y.'

We have defined a set of five distinct postures, each represented by **Boolean values**. In this scheme, each specific posture is assigned a value of **1** when actively performed by the user and **0** when the user is not in that specific posture.

To ensure the smooth operation of controlling the robot using the navigation stack, it is essential to incorporate postures as one of the input parameters. To facilitate this, we have developed a series of mathematical equations, with each equation corresponding to one of the aforementioned postures.

The following postures, each with its respective function precisely defined with mathematical equation, have been implemented on the Butler Robot:

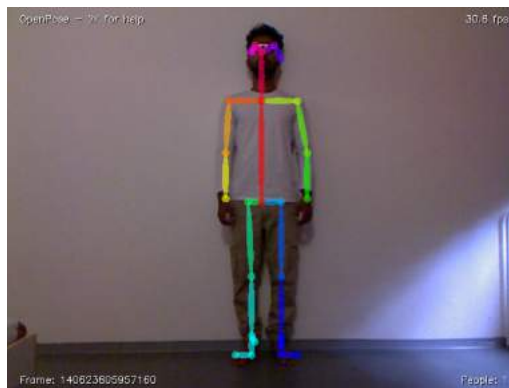
- **Facing Towards:** The individual directs their full attention to the robot while maintaining a forward-facing position.
- **Facing Backwards:** The person is oriented in the opposite direction, showing no attention or awareness towards the robot.
- **Facing Sideways:** The individual is positioned in a manner where they may or may not be aware of the robot, as they are not directly facing it.
- **Sitting:** The person is seated.
- **Standing:** The individual is in an upright, standing position.

Complementing these mathematical expressions are accompanying images that depict various poses, enhancing comprehension through visual aids.

$$(4.9) \text{ facing\_towards} = ((Nose_x) \times (LEye_x) \times (REye_x) \times (REar_x) \times (LEar_x))! = 0$$

The critical observation to highlight is that when specific coordinates possess a value of zero, it signifies the absence or non-visibility of a particular keypoint on the human subject. This principle underpins the formulation presented above. In the context of this formula, the absence of any coordinate, specifically focusing on the x-coordinate of the Nose and Ear keypoints, indicates that the individual is not directing their full attention towards the robot.

The provided Figure 4.16 offers valuable insights into the subject matter, providing a comprehensive visual representation of the concept.



**Figure 4.16:** Facing towards Robot

$$(4.10) \text{ facing\_backwards} = ((Nose_x = 0) \wedge (LEye_x = 0) \wedge (REye_x = 0))$$

If either the x or y coordinate value for the Nose, Left Eye, and Right Eye keypoints is zero, it signifies that the individual is oriented in the opposite direction with respect to the Robot.

The Figure 4.17 provides a comprehensive and visually informative illustration of the concept.



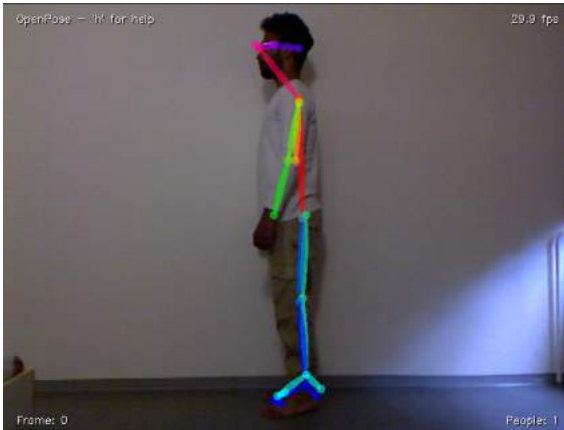
**Figure 4.17:** Facing backwards w.r.t Robot

$$(4.11) \text{ facing\_sideways} = (\text{Component\_A}) \vee (\text{Component\_B})$$

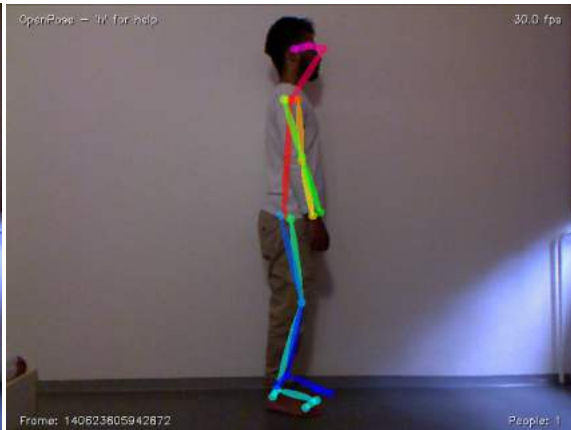
$$\text{Component\_A} = ((REar_x = 0) \wedge (LEar_C \geq Th_C))$$

$$\text{Component\_B} = ((LEar_x = 0) \wedge (REar_C \geq Th_C))$$

The Openpose library additionally provides a **confidence** parameter denoted as *Human\_Keypoint<sub>C</sub>* in above equations, serving as an indicator of the degree to which the respective intent has been accurately assigned to the human keypoints. Leveraging this parameter, we have adopted a selective approach, considering a specific keypoint only when its confidence level surpasses a predetermined threshold. This threshold, denoted as *Th<sub>C</sub>*, has been set at 0.6 in our software implementation, enhancing the determinism of our actions when interacting with the robot. Figure’s 4.18 and 4.19 gives a detailed and visually explanatory depiction of the concept.



**Figure 4.18:** Facing sideways 1<sup>st</sup> Posture



**Figure 4.19:** Facing sideways 2<sup>nd</sup> Posture

$$(4.12) \text{ sitting\_posture} = (\text{first\_criteria}) \vee (\text{second\_criteria})$$

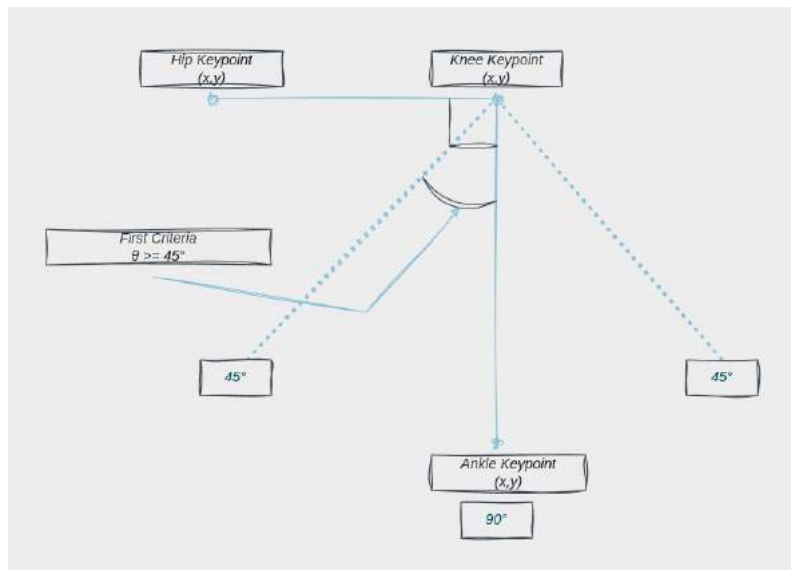
We have established two criteria for determining a sitting posture, each leading to a Boolean output of either one or zero, depending on whether the criteria are met:

- The **first criterion** involves the orientation of the individual relative to the robot. Specifically, when the person is facing sideways with respect to the robot, we assess the angle formed between the mid-hip and ankle in relation to the knee joint. In the given scenario, we define two slopes:
  1. “m1” represents the slope of the first line formed by connecting the hip and knee.
  2. “m2” denotes the slope of the second line formed by connecting the knee and ankle.
  3. To ensure that we measure the angle in a positive manner, we utilize the absolute value function, represented as “abs()”. This is crucial in calculating the angle accurately.

4. We then employ the arctangent function, denoted as “arctan()”, to compute the angle in radians.
5. The resulting angle, denoted as  $\theta$  signifies the measurement between these two lines and is expressed in degrees.

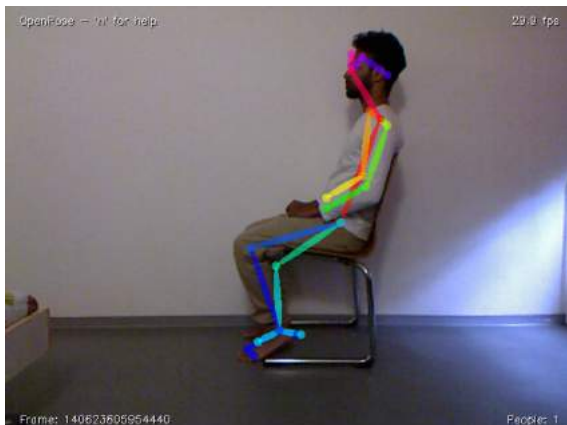
$$\theta = \arctan\left(\frac{|m_2 - m_1|}{1 + m_1 \cdot m_2}\right)$$

When the resultant angle, denoted as  $\theta$ , attains a value equal to or exceeding  $45^\circ$ , it fulfills the first criterion, classifying the specific human posture as sitting.



**Figure 4.20:** First Criteria : Representation of  $\theta$

The diagram 4.20 provides a conceptual representation of a mathematical expression, while the accompanying Figures 4.21 and 4.22 serve to illustrate specific postures or configurations associated with the concept.



**Figure 4.21:** Sitting 1<sup>st</sup> Posture



**Figure 4.22:** Sitting 2<sup>nd</sup> Posture

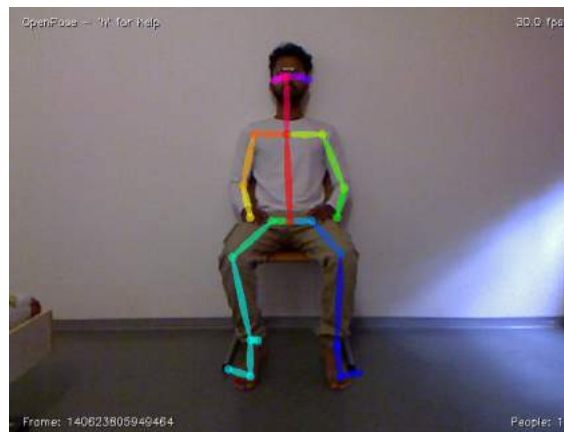
- The **second criterion** pertains to the person facing away from the robot. In this scenario, we examine the distance between the hip and knee joints. In our effort to compute the hip-to-knee distance, we have opted to exclusively take into account the vertical (y) coordinates. After conducting a series of iterative experiments at varying distances from the camera, we have ascertained that the distance in question must not exceed 15 units.

$$\text{right\_side\_difference} = |(RHip_y - RKnee_y)| \times 100$$

$$\text{left\_side\_difference} = |(LHip_y - LKnee_y)| \times 100$$

$$(4.13) \quad (\text{right\_side\_difference}) \leq 15 \text{ and } (\text{right\_side\_difference}) \leq 15$$

Figure 4.23 gives a detailed and visually explanatory depiction of the concept.



**Figure 4.23:** Sitting 3<sup>rd</sup> Posture

If either of these criteria is satisfied, the human's posture is classified as sitting, resulting in a Boolean value of one otherwise, it remains at zero.

$$(4.14) \quad \text{standing\_posture} = \neg(\text{standing\_posture})$$

For the sake of simplicity, we adopt the presumption that if a person does not meet the criteria for a sitting posture, then he or she is considered to be in a standing position by default.

### 4.3.3 Custom ROS message

In our implementation, it is imperative to incorporate gesture and posture information into the ROS environment. This objective is met by publishing a ROS topic through the use of a **custom ROS message**. Within the context of our specific use case, we have harnessed this functionality to define a ROS topic known as “/pose\_face”. This abbreviated term encapsulates the concept of human posture and facing orientation. The creation of this ROS topic is achieved by implementing a custom ROS message, as outlined in the table 4.1:

**Table 4.1:** PostureGesture.msg ROS Custom Message Structure

Field	Description
person_id	An integer representing the person's ID
pose	A string describing the Human pose
facing	A string describing the Human facing direction
gesture	A string describing the Human gesture

## 4.4 Social Rule's

We have established several safety criteria crucial for Human-Robot Interaction HRI.

- Firstly, we have delineated two distinct categories of gestures. These gestures serve as cues for the robot to discern whether it should comply with or disregard human directives. This categorization enhances the robot's capacity to interpret human actions and remain acutely attuned to its surrounding environment.
- The second criterion pertains to the maintenance of a safe separation distance between the human and the robot. This prescribed distance ranges from one meter to less than five meters. When the human approaches too closely to the robot, the latter refrains from responding to any human gestures. This safety measure ensures that the robot maintains a buffer zone to mitigate potential physical contact.
- The third criterion necessitates the entire human body to be within the camera frame for the robot to engage with user gestures effectively. Until the entirety of the user's body is visible within the frame, the robot refrains from responding. This prerequisite ensures that the robot possesses a comprehensive view of the scenario, allowing it to act in accordance with a better understanding of the situation.
- The fourth and final criterion mandates that the human should be oriented toward the robot. Specifically, the robot will only execute user commands when the human is directing their full attention towards it. In situations where the human's focus is divided, the robot remains in a state of readiness, awaiting undivided attention from the human to ensure that it responds accurately to their actions. This criterion underscores the importance of the human's undivided attention for effective HRI.



## 4.5 Autonomous Navigation

Our initial concept for our use case involved the control of robot movement through gestures. However, an additional challenge we faced was the need for dynamic obstacle avoidance, which we recognized could be achieved by integrating SLAM with a Navigation stack. Initially, we explored two primary ideas.

- The first idea centered on utilizing depth information from the camera to facilitate obstacle avoidance by providing appropriate velocity and angular commands to the robot.
- The second idea involved the implementation of one of the BUG algorithms, specifically choosing from BUG0, BUG1, and BUG2. However, upon closer examination, both of these approaches were found to be lacking in efficiency and robustness.

Another important factor to note is that the camera was not affixed to the robot itself, which introduced a degree of complexity to the implementation of the aforementioned concepts. Subsequently, we shifted our focus towards creating an environment map with Hector SLAM and refining the navigation stack to enhance the overall robustness and efficiency of our system.

Numerous changes and enhancements have been implemented in comparison to the previous efforts by Arfa [2] and Avinash Kangare [3]. These alterations have significantly bolstered the robot's navigation capabilities and dynamic obstacle avoidance, rendering them markedly more robust and efficient.

### 4.5.1 Environment Mapping - Hector SLAM

The necessity arose to create a fresh environmental map due to substantial alterations in the office layout, which introduced new obstacles into the environment. When considering the creation of an environment map for our research, we encountered two viable options. The first option was to utilize Hector SLAM, which had previously been implemented and was readily available as a base. The second option involved the implementation of a more efficient SLAM algorithm, namely Cartographer SLAM. However, due to the constraints imposed by limited time and the inherent complexity of Cartographer SLAM [25], its practical implementation posed a significant challenge [66].

To make an informed decision, we conducted thorough research to compare various SLAM implementations, weighing their respective advantages and disadvantages [17]. Following a comprehensive analysis and considering the time factor, we ultimately opted to proceed with Hector SLAM. This choice was driven by the fact that Hector SLAM consistently provided accurate mapping results, particularly well-suited for indoor environments - a crucial aspect of our research's requirements [34].

The earlier implementations carried out by Arfa [2] and Avinash Kangare [3] provide comprehensive insights and concepts related to Hector SLAM. However, in this research study, our focus is directed solely towards highlighting the critical parameters that have significantly enhanced the implementation's efficiency and robustness.

The key subtle refinements made were as follows:

- The repositioning of the Lidar sensor on Robot to optimize obstacle detection by utilizing a 2D Lidar system, thereby enhancing its scanning plane coverage.
- These modifications were also synchronized with adjustments in the URDF file, ensuring the accurate placement of the Lidar sensor. This rectification effectively resolved a previously unresolved issue that had persisted in earlier implementations. The Figure 4.25, along with the modification in the URDF file 4.24, effectively communicates the essence of the significant alteration.

```

<link name="laser_frame">
<inertial>
<origin xyz="0.00029774 0.0006667 0.00013047" rpy="0 0 0" />
<mass value="0.15717" />
<inertia ixx="6.7885E-05" ixy="-1.3987E-07" ixz="-8.1554E-07"
iyy="0.00013173" iyz="-9.0932E-08" izz="7.1972E-05" />
</inertial>
<visual>
<geometry>
<cylinder radius="0.05" length="0.04" />
<!--mesh
| | | filename="package://tb_description/meshes/ydlidar.dae" /-->
</geometry>
<material name="">
<color rgba="0.64706 0.61961 0.58824 1" />
</material>
</visual>
<collision>
<origin xyz="0 0 0" rpy="0 0 0" />
<geometry>
<cylinder radius="0.05" length="0.04" />
<!--mesh
| | | filename="package://ydlidar/meshes/ydlidar.dae" /-->
</geometry>
</collision>
</link>
<joint name="laser_range_mount_joint" type="fixed">
<origin rpy="0 0 0" xyz="-0.05 0 0.05" />
<parent link="base_link" />
<child link="laser_frame" />
</joint>

```

Figure 4.24: Ohmni Robot URDF file.

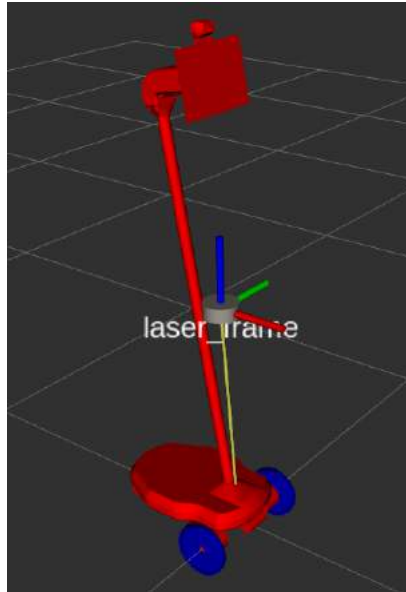


Figure 4.25: Position of LiDAR in URDF file.

We embarked on the task of implementing environment mapping from the ground up, taking the time to grasp the underlying concepts and applying them practically. Several changes were introduced in the software launch files to facilitate this process, drawing insights from the [56]:

- In `SLAM_ws/src/hector_slam/hector_mapping/launch/mapping_default.launch`:

1. A node was incorporated to handle static data transformation:

```

<node pkg="tf" type="static_transform_publisher"
name="base_to_laser_broadcaster" args="0 0 0 0 0 0 base_link laser 100"/>

```

2. An argument for specifying the base frame was included:

```

<arg name="base_frame" default="base_link"/>

```

3. An argument for specifying the base frame was included:

```

<arg name="odom_frame" default="base_link"/>

```

- In `SLAM_ws/src/hector_slam/hector_slam_launch/launch/tutorial.launch`:

1. An inclusion of a ROS driver for YD Lidar was added, which is launched using the following line::

```
<include file="$(find ydlidar_ros_driver)/launch/ydlidar.launch"/>
```

2. The parameter for using simulated time was set to **'false'**:

```
<param name="/use_sim_time" value="false"/>
```

These changes serve to enhance the functionality and accuracy of the environment mapping implementation, ensuring that the necessary components and configurations are in place for successful execution.

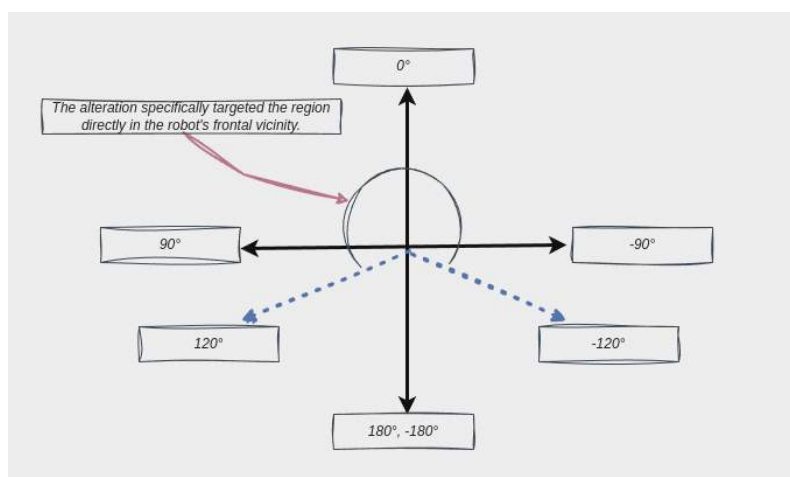
Within the launch file of the YD Lidar ROS driver, a significant modification was made concerning the region considered during the creation of the environmental map. Specifically, only the plane directly in front of the robot was taken into account. This decision was informed by the author's positioning behind the robot to facilitate its movement within the environment while simultaneously mapping it. Notably, since Hector SLAM operates independently of the `/odom` frame, the mapping process was executed without necessitating the consideration of `tf` (transform) data from the robot. In this scenario, the Lidar was powered and positioned on the robot during the mapping of the environment.

- In `SLAM_ws/src/ydlidar_ros_driver/launch/ydlidar.launch`:

```
<param name="angle_min" type="double" value="-120" />
```

```
<param name="angle_max" type="double" value="120" />
```

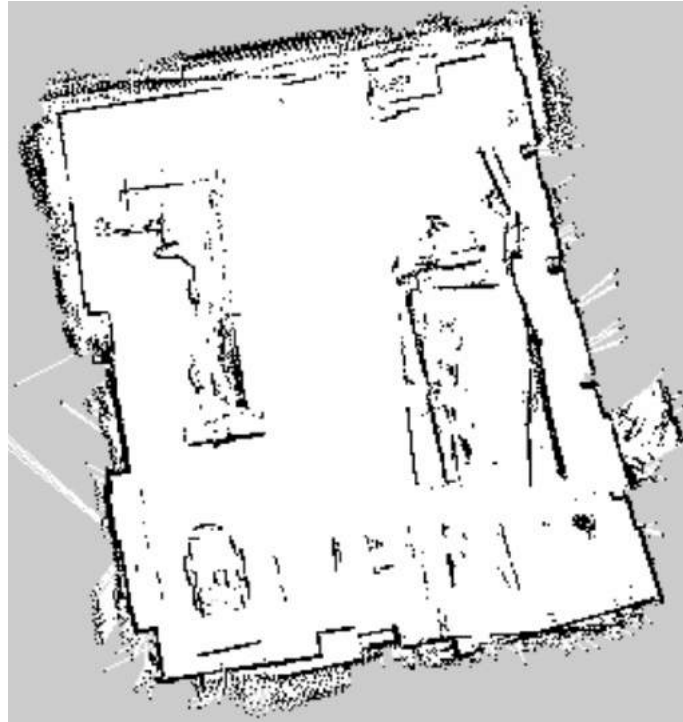
The adjustments reflect a narrower angle range considered during the mapping process, which spans from -120 degrees to 120 degrees.



**Figure 4.26:** YDLIDAR X4 coordinates definition

The Figure 4.26 provides insight into the angles taken into account during the environment mapping process, spanning from -120 to 120 degrees.

After launching the required files and preserving the map 4.27 generated by Hector SLAM, we successfully generated the following representation of the environment.



**Figure 4.27:** Map generated by Hector SLAM

### 4.5.2 Navigation Stack - Butler

Obtaining a thorough understanding of the human follower use case, where the navigation stack played a crucial role, was imperative. We referenced [3] and consulted pertinent papers to gain valuable insights into its implementation [4] [71]. Now, our research efforts were primarily centered on the field of path planning and its significance within the realm of HRI. Path planning constitutes a foundational element of autonomous navigation, a fundamental aspect of HRI systems. By carefully examining the methodologies and findings presented in various research studies [58], our aim was to acquire a profound comprehension of effective path planning strategies and their practical applications in HRI scenarios Aboki et al. [1]. The knowledge acquired through this extensive research laid a solid foundation for our research and also equipped us with the necessary insights to begin our collaboration with the Butler Robot [30]. It allowed us to approach the integration of our use case with a robust and informed perspective, optimizing the chances of a successful implementation [65].

Managing a robot's Navigation Stack poses a considerable challenge. Given our limited time-frame, conducting an exhaustive examination of each parameter was not feasible. Although the navigation stack had been previously implemented by our colleagues, it lacked the robustness and efficiency

required for seamless integration with our Gesture and Posture research. It's essential to emphasize that each parameter, both in the configuration files and various launch files of the navigation stack, plays a pivotal role and has a substantial impact on the success of autonomous navigation.

Recognizing the complexity of previous implementations, we made the deliberate choice to adopt a simpler and more methodical approach, starting from the basics to ensure a better understanding. Our work with the navigation stack commenced with the imperative need to acquaint ourselves with its detail workings. This involved studying comprehensive documentation and reviewing its practical application on other robotic platforms. To establish a solid foundation, we turned to ROS tutorials [Navigation Stack Setup](#) [33] and *ROS Navigation Stack* [54], which provided essential insights into the fundamentals of the Navigation Stack. Additionally, these tutorials served as a valuable reference for fine-tuning the navigation stack parameters. In our pursuit of a more profound understanding of the navigation stack, Zheng [73] authored navigation guide assumed a crucial role. It served as a good resource, shedding light on the significance of diverse parameters and their vital interactions within the navigation system, thereby enhancing our knowledge substantially. For a practical, step-by-step implementation of the setup, the Husarion [20] tutorials proved to be an invaluable resource. Their clear explanations and systematic approach aided in a seamless setup process. It's worth noting that the work carried out by emanual robotis [45] exerted a significant influence on our own efforts. Their contributions served as a source of inspiration and guidance throughout our research.

Several noteworthy changes have been implemented in our current research compared to previous implementations:

- **Enhancements to File Structure:** Significant improvements were made to the file structure, resulting in a more organized and readable workflow.
- **Parameter Refinement:** Corrected misdefined parameters and eliminated unnecessary ones, optimizing the overall configuration.
- **Navigation Stack Cleanup:** Removed redundant YAML files and selected ROS packages that were no longer required, streamlining the Navigation Stack.

The primary objective was to assess and experiment with various navigation stacks on the Butler Robot, drawing insights from their successful implementation on other robotic platforms. This approach was underpinned by a solid understanding of the robot's specific requirements, purpose, and utility. Through a process of trial and error, coupled with strategic adjustments, we arrived at an optimal set of parameters for our navigation stack. These refinements significantly enhanced the capabilities of navigation and dynamic obstacle avoidance in comparison to previous implementations. Notably, the navigation parameters employed in the **Turtle Bot framework** "*Turtle Bot - Nav parameters* [61]" proved invaluable, streamlining our efforts and expediting the navigation stack setup a critical time-saving factor.

The navigation stack relies on a specific robot configuration for its operation. The diagram 4.28 depicted above offers an overview of this requisite setup. In the diagram, the white components represent mandatory elements that have already been implemented, the gray components denote optional elements that are also already in place, and the blue components signify the necessity to create custom components tailored to each robot platform. The sections below provide a comprehensive outline of the prerequisites for the navigation stack, accompanied by detailed instructions on how to meet each requirement.

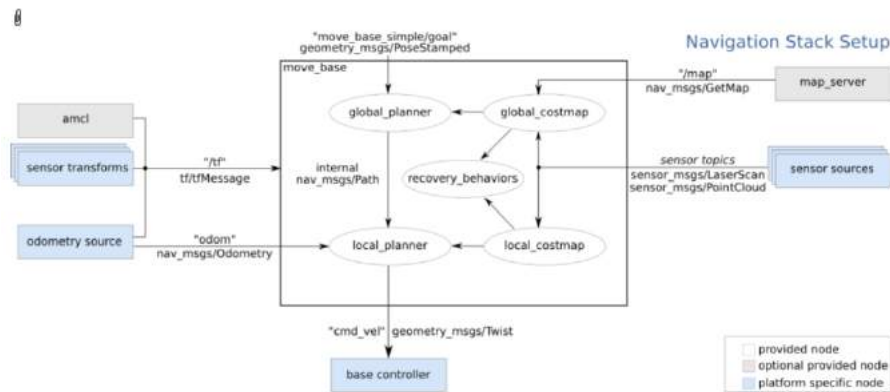


Figure 4.28: Navigation Stack Setup [ROS Wiki] [33]

Moving forward, our focus will be on the pivotal parameters outlined in the configuration files, as well as key components featured in select launch files. These adjustments collectively contribute to an impressive realization of autonomous navigation.

#### 4.5.2.1 ohmni\_2dnv Package

We've developed a dedicated ROS package to house and manage all the configuration and launch files associated with the navigation stack. This package relies on the move\_base package, which provides the high-level interface for the navigation stack.

#### 4.5.2.2 Butler Configuration Launch File

The `ohmni_bring_up.launch` is a roslaunch file responsible for initializing all the necessary hardware components and publishing the required transforms for the robot's operation.

```

ohmni_bring_up.launch X
src > ohmni_2dnv > src > launch > ohmni_bring_up.launch
1 <?xml version="1.0"?>
2 <launch>
3
4 <!-- Invoke YD lidar -->
5 <!-- check the min_max angle in YD lidar launch file -->
6 <include file="$(find ydlidar_ros_driver)/launch/ydlidar.launch"/>
7
8 <!-- send urdf to param server, used by robot state publisher and joint state publisher -->
9 <param name="robot_description" command="$(find xacro)/xacro --inorder '$(find ohmni_2dnv)/src/urdf/ohmni_bot.urdf.xacro' />
10
11 <node name="joint_state_publisher" pkg="joint_state_publisher" type="joint_state_publisher">
12 | <!-- param name="use_gui" value="false"/ -->
13 </node>
14
15 <!-- Send robot states to tf -->
16 <node name="robot_state_publisher" pkg="robot_state_publisher" type="robot_state_publisher" respawn="false" output="screen"/>
17
18 </launch>

```

Figure 4.29: ohmni\_bring\_up.launch file

Launch file snippet 4.29 serves the purpose of configuring several essential components for the robot's hardware setup. It encompasses tasks such as initializing the YD Lidar sensor, defining the robot's URDF model, and launching crucial nodes responsible for joint state publication and the transformation of the robot's state information.

### 4.5.2.3 Costmap Configuration

The navigation stack employs two distinct costmaps to retain information concerning obstacles within the robot's environment. One costmap serves the purpose of global planning, facilitating the creation of long-term plans across the entire spatial domain. The other is dedicated to local planning and real-time obstacle avoidance. To address the configuration needs effectively, we categorize them into three sections: common configuration options, which apply universally to both costmaps, global configuration options specific to the global planning costmap, and local configuration options tailored to the local planning and obstacle avoidance costmap. This approach ensures precise configuration control for each costmap while accommodating shared settings when necessary [Navigation Stack Setup] [46]. To access comprehensive documentation encompassing the entire spectrum of available options, we recommend referring to the `costmap_2d` [46] documentation.

1. **Common Configuration:** The navigation stack utilizes costmaps as foundational data structures for the storage and organization of obstacle-related information within the robot's operational surroundings. These costmaps adopt a grid-based representation of the environment, wherein each grid cell contains pertinent data regarding the navigational cost linked to its particular location. As crucial elements in the robot's decision-making framework, these costmaps furnish vital spatial information that proves instrumental in tasks such as path planning, localization, and obstacle avoidance.

```

costmap_common_params.yaml x
src > ohmni_2dnav > src > config > costmap_common_params.yaml
1
2 #http://wiki.ros.org/move_base
3 #https://husarion.com/tutorials/ros-tutorials/7-path-planning/
4
5 obstacle_range: 3.0 #In this range obstacles will be considered during path planning.
6 raytrace_range: 3.5 #This parameter defines range in which area could be considered as free.
7
8 footprint: [[-0.40, -0.20], [-0.40, 0.20], [0.077, 0.20], [0.077, -0.20]] #this parameter defines coordinates of Ohmni robots outline,
9 #this will be considered during collision detecting, if the robot is
# not circular
10 #robot radius: 0.17 # distance a circular robot should be clear of the obstacle
11 #center point considered for footprint is -> robot_base_frame: footprint
12
13 inflation_radius: 0.5 #this parameter defines distance to obstacle where cost should be considered,
14 #any further from obstacle than this value will be treated as no cost.
15 cost_scaling_factor: 3.0
16
17 map_type: costmap
18 observation_sources: scan
19 scan: {sensor_frame: laser_frame, data_type: LaserScan, topic: scan, marking: true, clearing: true}

```

Figure 4.30: Common configuration file

2. **Global Configuration:** Global Configuration parameters are essential for configuring the global costmap used in robot navigation, with details about update frequencies, reference frames, and whether the map is considered static or not.

```

global_costmap_params.yaml x
src > ohmni_2dnav > src > config > global_costmap_params.yaml
1 global_costmap:
2   global_frame: map
3   robot_base_frame: footprint
4
5   update_frequency: 10.0
6   publish_frequency: 10.0
7   transform_tolerance: 0.5
8
9   static_map: true #Map remains static for global path

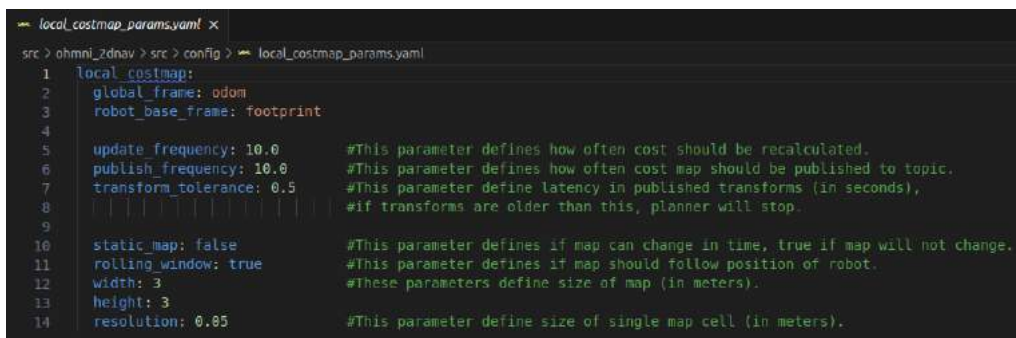
```

Figure 4.31: Global costmap parameters configuration file



“global\_frame” sets the reference frame for this costmap as “map”, and “robot\_base\_frame” defines the robot’s base frame. We control the update and publication frequencies with “update\_frequency” and “publish\_frequency” both set to 10 Hz. To ensure accurate frame transformations, we use “transform\_tolerance” with a tolerance of 0.5 seconds. Finally, by setting “static\_map” to true, we indicate that the global map remains unchanged during navigation, assuming a static environment.

3. **Local Configuration:** These parameters pertain to the local costmap configuration for robot navigation. The “local\_costmap” section specifies settings related to short-term path planning. “global\_frame” sets the reference frame for this costmap as “odom”, and “robot\_base\_frame” defines the robot’s footprint frame. The update and publication frequencies are controlled by “update\_frequency” and “publish\_frequency”, both set to 10 Hz. “transform\_tolerance” determines the allowed latency in published transformations, with a threshold of 0.5 seconds. The “static\_map” parameter is set to “false”, indicating that the map can change over time. “rolling\_window” is set to “true”, enabling the costmap to move with the robot’s position. Lastly, “width”, “height”, and “resolution” define the size and granularity of the costmap in meters. These parameters collectively configure the local costmap for agile and responsive robot navigation.



```

local_costmap_params.yaml x
src > ohmini_2dnav > src > config > local_costmap_params.yaml
1 local_costmap:
2   global_frame: odom
3   robot_base_frame: footprint
4
5   update_frequency: 10.0 #This parameter defines how often cost should be recalculated.
6   publish_frequency: 10.0 #This parameter defines how often cost map should be published to topic.
7   transform_tolerance: 0.5 #This parameter define latency in published transforms (in seconds),
8                             #if transforms are older than this, planner will stop.
9
10  static_map: false #This parameter defines if map can change in time, true if map will not change.
11  rolling_window: true #This parameter defines if map should follow position of robot.
12  width: 3 #These parameters define size of map (in meters).
13  height: 3
14  resolution: 0.85 #This parameter define size of single map cell (in meters).

```

Figure 4.32: Local costmap parameters configuration file

4. **DWA Local Planner Configuration:** The robot’s velocity limits are set using parameters like “max\_vel\_x” and “min\_vel\_x” for forward and backward motion, “max\_vel\_theta” and “min\_vel\_theta” for rotational motion, and “max\_vel\_trans” and “min\_vel\_trans” for straight-line movement. Additionally, acceleration limits are specified with “acc\_lim\_x” and “acc\_lim\_theta”. Goal tolerance parameters, such as “xy\_goal\_tolerance” and “yaw\_goal\_tolerance”, determine how close the robot needs to be to its target pose to consider it reached. The configuration also defines parameters related to trajectory scoring, such as how the planner weights sticking to the global path (“path\_distance\_bias”) and avoiding obstacles (“occdist\_scale”). Parameters like “forward\_point\_distance” and “scaling\_speed” influence the robot’s behavior when following trajectories. To prevent oscillations, the planner uses “oscillation\_reset\_dist” to reset flags when the robot travels a certain distance without significant progress. Finally, for debugging purposes, it can publish trajectory and cost grid information. These parameters collectively tailor the DWAPlanerROS to the specific dynamics and goals of the robot, ensuring effective navigation while adhering to velocity and acceleration constraints, avoiding obstacles, and reaching the desired target pose.



```

src > ohmni_2dnav > src > config > dwa_local_planner_params.yaml
1 DWAPlannerROS:
2
3 # Robot Configuration Parameters
4 max_vel_x: 0.10
5 min_vel_x: -0.10
6
7 max_vel_y: 0.0
8 min_vel_y: 0.0
9
10 # The velocity when robot is moving in a straight line
11 max_vel_trans: 0.10
12 min_vel_trans: 0.09
13
14 max_vel_theta: 0.0
15 min_vel_theta: -0.8
16
17 acc_lim_x: 1.5
18 acc_lim_y: 0.0
19 acc_lim_theta: 2.0
20
21 # Goal Tolerance Parameters
22 xy_goal_tolerance: 0.05
23 yaw_goal_tolerance: 0.17
24 latch_xy_goal_tolerance: false
25
26 # Forward Simulation Parameters
27 sim_time: 2.0
28 vx_samples: 20
29 vy_samples: 0
30 vth_samples: 40
31 controller_frequency: 10.0
32
33 # Trajectory Scoring Parameters
34 path_distance_bias: 32.0 # 32.0 - weighting for how much it should stick to the global path plan
35 goal_distance_bias: 20.0 # 20.0 - weighting for how much it should attempt to reach its goal
36 gcdfx1: scale: 0.02 # 0.02 - weighting for how much the controller should avoid obstacles
37 forward_point_distance: 0.325 # 0.325 - how far along to place an additional scoring point
38 stop_time_buffer: 0.2 # 0.2 - amount of time a robot must stop in before colliding for a valid trail
39 scaling_speed: 0.25 # 0.25 - absolute velocity at which to start scaling the robot's footprint
40 max_scaling_factor: 0.2 # 0.2 - how much to scale the robot's footprint when at speed.
41
42 # Oscillation Prevention Parameters
43 oscillation_reset_dist: 0.05 # 0.05 - how far to travel before resetting oscillation flags
44
45 # Debugging
46 publish_traj_pc: true
47 publish_cost_grid_pc: true

```

Figure 4.33: DWA Local Planner parameters configuration file

5. **Move Base Configuration:** In the Figure 4.34, you will find additional parameters specified within the move\_base configuration file. The comments provided within the configuration file offer comprehensive explanations for each of these parameters.

```

src > ohmni_2dnav > src > config > move_base_params.yaml
1 shutdown_costmap: false #Determines whether or not to shutdown the costmaps of the node when move_base is in an inactive state
2 controller_frequency: 10.0 #The rate in Hz at which to run the control loop and send velocity commands to the base
3 planner_patience: 3.0 #How long the planner will wait in seconds in an attempt to find a valid plan before space-clearing operations are performed.
4 controller_patience: 15.0 #How long the controller will wait in seconds without receiving a valid control before space-clearing operations are performed.
5 conservative_reset_dist: 3.0 #The distance away from the robot in meters beyond which obstacles will be cleared from the costmap when attempting to clear space in the map. Note, this parameter is only used when the default recovery behaviors are used for move_base.
6 planner_frequency: 5.0 #The rate in Hz at which to run the global planning loop.
7 oscillation_timeout: 10.0 #How long in seconds to allow for oscillation before executing recovery behaviors. A value of 0.0 corresponds to an infinite timeout.
8 oscillation_distance: 0.2 #How far in meters the robot must move to be considered not to be oscillating.

```

Figure 4.34: Move Base parameters configuration file

6. **AMCL Configuration:** In this context, it is worth noting that the **initial pose** of the robot on the map can be specified within the AMCL launch file by utilizing parameters, as illustrated in Figure 4.35. For a comprehensive understanding of each parameter's intricacies, we recommend consulting the ROS documentation available for reference [AMCL] [47].
7. **Launch File for the Navigation Stack:** The provided ROS launch file snippet 4.36 plays a crucial role in orchestrating the navigation stack for a mobile robot. It begins by including the 'ohmni\_bring\_up.launch' file, which is responsible for initializing the robot's essential components, including hardware drivers, sensors, and communication interfaces. Following that, an argument named 'map\_file' is defined, pointing to a YAML map file ('lab\_map\_2.yaml') in the 'ohmni\_2dnav' package, which serves as the representation of

```
</> amcl.launch x
src > ohmni_2dnav > src > launch > </> amcl.launch
6
7 <!--Added the initial pose of the robot -->
8 <arg name="initial_pose_x" default="-1.178"/>
9 <arg name="initial_pose_y" default="0.996"/>
10 <arg name="initial_pose_a" default="-3.001"/>
11
```

Figure 4.35: AMCL configuration parameters

the robot's environment. The next step involves starting the 'map\_server' node from the 'map\_server' package, which provides map data to the navigation system using the specified 'map\_file' argument. Additionally, the launch file includes 'amcl.launch', configuring and launching the AMCL (Adaptive Monte Carlo Localization) package for accurate localization within the map. Finally, it includes 'move\_base.launch', which sets up the critical 'move\_base' package responsible for path planning and obstacle avoidance during the robot's navigation tasks. Altogether, this launch file streamlines the setup and coordination of essential components, enabling the robot to navigate autonomously within its environment.

```
</> ohmni_nav.launch x
src > ohmni_2dnav > src > launch > </> ohmni_nav.launch
1 <?xml version="1.0"?>
2 <launch>
3
4 <!-- Bring up the ROBOT -->
5 <include file="$(find ohmni_2dnav)/src/launch/ohmni_bring_up.launch"/>
6
7 <!-- Map server -->
8 <arg name="map_file" default="$(find ohmni_2dnav)/src/maps/lab_map_2.yaml"/>
9 <node name="map_server" pkg="map_server" type="map_server" args="$(arg map_file)" />
10
11 <!-- start the AMCL package -->
12 <include file="$(find ohmni_2dnav)/src/launch/amcl.launch"/>
13
14 <!-- move_base -->
15 <include file="$(find ohmni_2dnav)/src/launch/move_base.launch"/>
16
17 <!-- run rviz seperately-->
18 <!--node pkg="rviz" name="rviz" type="rviz" args="-d $(find ohmni_2dnav)/src/config/rvizconfig.rviz" /-->
19
20 </launch>
```

Figure 4.36: Launch File for the Navigation Stack

## 5 Realization of Solution

### 5.1 ROS Driver for Kinect Depth Camera

In our research project, we utilized two distinct ROS drivers for the Kinect Xbox Camera: namely, `openni_launch` and `freenect_launch`. Both of these ROS drivers exhibited seamless compatibility when operating within the Ubuntu 20.04 LTS Operating System and the ROS Noetic Distribution. However, it's noteworthy that our Docker image, which has been specifically configured for use with the Ubuntu 18.04 LTS Operating System and the Melodic ROS distribution, exclusively supports the `freenect_launch` driver [49] [48].

Comprehensive documentation is readily accessible on the following websites,

1. [openni\\_launch camera ROS driver](#) [49]
2. [freenect\\_launch camera ROS driver](#) [48]

providing detailed instructions on the utilization of the ROS driver, complete with tutorials and clear explanations of the arguments employed in the launch file.

In our study, we exclusively employed the `freenect_launch` ROS driver, which can be initiated using the following command:

```
roslaunch freenect_launch freenect.launch
```

`freenect_launch` camera ROS driver provides access to several key ROS topics, which include:

- `/camera/rgb/image_raw`: This topic offers the RGB image data captured by the Kinect Camera.
- `/camera/depth_registered/image_raw`: It provides access to the depth image information obtained from the Kinect Camera.
- `/camera/rgb/camera_info`: This topic is essential for retrieving the camera's calibration parameters.
- `camera_depth_optical_frame`: This topic specifies the tf frame ID associated with the IR camera.

### 5.1.1 Camera Calibration

In the calibration process for the Kinect camera, we followed a calibration procedure for both the RGB camera and the IR camera responsible for providing depth information. While the RGB camera calibration was successfully completed, we encountered challenges during the calibration of the IR camera, encountering a similar issue as previously encountered [Problem with IR image](#). The camera calibration process was conducted by following the tutorial - [Calibration of the Kinect camera](#) [6]

The following commands were employed for the camera calibration process:

- RGB Camera :

```
roslaunch camera_calibration cameracalibrator.py image:=/camera/rgb/image_raw
camera:=/camera/rgb --size 10x7 --square 0.024
```

- IR Camera :

```
roslaunch camera_calibration cameracalibrator.py image:=/camera/ir/image_raw camera:=/camera/ir
--size 10x7 --square 0.024
```

As the issue remained unresolved despite our efforts, we opted to utilize the default camera calibration intrinsic parameters. Through experimentation, we determined that the error in the depth information was negligible under these default settings.

## 5.2 Docker Image - Openpose Library

We encountered numerous challenges while attempting to build the Openpose library [Openpose library](#) [7] on an Ubuntu 20.04 LTS operating system. Even after successfully compiling the library, we encountered integration issues when incorporating it into the Openpose ROS wrapper [ROS Openpose Wrapper](#) [21]. Consequently, we devised a solution involving the utilization of Docker for our implementation and proceeded with this approach. Extensive research revealed that the Openpose library has undergone more rigorous testing on hardware running Ubuntu 18.04 LTS. As a result, we made the strategic decision to utilize Ubuntu 18.04 as the base image for our Docker container. This image was thoughtfully configured to include the ROS Melodic distribution, the Openpose library, and all essential dependencies, including CUDA and CuDNN. The resulting Docker image was meticulously built and proved to be a highly functional environment, where both the Openpose library and ROS Melodic Distribution operated seamlessly within the container instance [15].

### 5.2.1 NVIDIA Container Toolkit and Configuring Docker.

Prior to commencing work with the Openpose Docker Image, it is imperative to emphasize the significance of installing the **NVIDIA Container Toolkit and Configuring Docker**. This toolkit enables the Docker container to effectively utilize NVIDIA GPUs, enhancing computational capabilities and performance. For the installation process, the following link provides comprehensive guidance [Installing the NVIDIA Container Toolkit and Configuring Docker](#) [35].

## 5.2.2 Openpose Docker file

The provided snippet furnishes insight into the content encapsulated within our Docker file. It offers a preview of the Docker file's composition and the elements it contains. Here is the GitHub link to access the Docker file : [Openpose Docker file](#)

```

1
2
3 # <https://hub.docker.com/r/cwaffles/openpose>
4 FROM nvidia/cuda:10.1-cudnn7-devel-ubuntu18.04
5
6
7 #get deps
8 RUN apt-get update && \
9 DEBIAN_FRONTEND=noninteractive apt-get install -y --no-install-recommends \
10 python3-dev python3-pip python3-setuptools git g++ wget make libprotobuf-dev protobuf-compiler libopencv-dev \
11 libgoogle-glog-dev libboost-all-dev libcaffe-cuda-dev libhdf5-dev libatlas-base-dev
12
13
14 #for python api
15 #RUN pip3 install scikit-build
16 RUN pip3 install scikit-build numpy opencv-python==4.1.1.26
17
18
19 #replace cmake as old version has CUDA variable bugs
20 RUN wget https://github.com/Kitware/CMake/releases/download/v3.16.0/cmake-3.16.0-Linux-x86_64.tar.gz && \
21 tar xzf cmake-3.16.0-Linux-x86_64.tar.gz -C /opt && \
22 rm cmake-3.16.0-Linux-x86_64.tar.gz
23 ENV PATH="/opt/cmake-3.16.0-Linux-x86_64/bin:${PATH}"
24
25
26 #ADDED PORTION IN FILE :STARTS
27 # Install ROS Melodic
28 RUN apt-get update && \
29 DEBIAN_FRONTEND=noninteractive apt-get install -y --no-install-recommends \
30 apt-utils \
31 dirmngr \
32 gnupg2 \
33 lsb-release
34 RUN apt-key adv --keyserver keyserver.ubuntu.com --recv-keys C1CF6E31E6BADE8868B172B4F42ED6FBAB17C654
35 RUN echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list
36 RUN apt-get update && \
37 DEBIAN_FRONTEND=noninteractive apt-get install -y --no-install-recommends \
38 ros-melodic-desktop-full
39
40 #END OF PORTION
41
42
43 #get openpose
44 WORKDIR /openpose
45 RUN git clone -q --depth 1 https://github.com/CMU-Perceptual-Computing-Lab/openpose.git .
46
47
48 #build it
49 WORKDIR /openpose/build
50 RUN cmake -DBUILD_PYTHON=ON \
51     -DDOWNLOAD_BODY_25_MODEL=ON \
52     -DDOWNLOAD_BODY_MPT_MODEL=OFF \
53     -DDOWNLOAD_HAND_MODEL=OFF \
54     -DDOWNLOAD_FACE_MODEL=OFF \
55     ..
56
57 #these commands are commenting out a line of code in two CMake files named Cuda.cmake
58 #by replacing the line that sets the AMPERE variable with a commented version of the same line.
59 RUN sed -ie 's/set(AMPERE "80 86")/#&g' ../cmake/Cuda.cmake && \
60     sed -ie 's/set(AMPERE "80 86")/#&g' ../3rdparty/caffe/cmake/Cuda.cmake
61 RUN make -j`nproc`
62 RUN make install
63 WORKDIR /openpose

```

Figure 5.1: Openpose Docker file

Within the Docker image instance, we initiated our work by incorporating supplementary software components, notably the ROS Openpose wrapper, a crucial element for seamless integration of the Openpose library within the ROS environment, and a ROS camera driver. To ensure the proper functioning of these components, necessary modifications were meticulously applied to the launch files of the ROS Openpose wrapper. These adjustments were executed with precision, resulting in the successful compilation and execution of the ROS Openpose wrapper within the Docker container. As a result, we achieved the capability to obtain precise 3D human poses, comprising 25 key body keypoints, a significant milestone in our research's development.

The Docker image instance, containing all locally applied modifications, was uploaded to the Docker Hub. This process involved the creation of an image encapsulating all the updates and installations performed on the local system within the Docker image instance. Subsequently, rigorous testing was conducted on various hardware platforms to validate the portability and universal compatibility of our Docker image. This comprehensive testing ensures that the Docker image is deployable and functional for users across diverse Linux systems, regardless of their specific hardware configurations. Here is the ultimate Docker image : [Openpose Docker Hub Image Patil \[40\]](#)

To interact with the Docker image, the initial step involves creating an instance that functions as a container. The following command is employed to retrieve the image from a local repository and execute it as a container:

```
localhost@shell:~$ sudo xhost +si:localuser:root
```

```
localhost@shell:~$ docker run --gpus all -e DISPLAY=$DISPLAY --env NVIDIA_VISIBLE_DEVICES=all
--env NVIDIA_DRIVER_CAPABILITIES=all --env DISPLAY --env QT_X11_NO_MITSHM=1
-v /tmp/.X11-unix:/tmp/.X11-unix -v /dev:/dev --net=host --privileged
--device=/dev/video0:/dev/video0 -it vin8/openpose_vin:final /bin/bash
```

Here is the explanation of arguments used in the above docker run command,

- The `sudo xhost +si:localuser:root` command allows the root user to access the current user's X11 display, which is necessary when running certain GUI applications with elevated privileges.
- The `-v` option is used to make directories or files from the host machine available inside the container.
- `docker run` is used to start a new Docker container from an image.
- `--gpus all` is used to make all available NVIDIA GPUs on the host machine accessible to the container.
- `-e DISPLAY=$DISPLAY` sets an environment variable in the container to the value of the `$DISPLAY` environment variable on the host machine. This is necessary to allow the container to display graphical output on the host machine's X11 display server.
- `--env NVIDIA_VISIBLE_DEVICES=all` sets an environment variable in the container to make all NVIDIA GPUs visible to OpenPose.
- `--env NVIDIA_DRIVER_CAPABILITIES=all` sets an environment variable in the container to specify that all driver capabilities are available.
- `--env DISPLAY` sets the `DISPLAY` environment variable in the container.
- `-v /tmp/.X11-unix:/tmp/.X11-unix` mounts the X11 UNIX socket directory on the host machine to the same directory in the container, which allows the container to access the host machine's X11 display server.

- `-v /dev:/dev` mounts the `/dev` directory on the host machine to the same directory in the container, which allows the container to access the host machine's devices.
- `--net=host` sets the network mode of the container to "host," which allows the container to access the host machine's network interfaces.
- `--privileged` gives the container full access to the host machine's system resources.
- `--device=/dev/video0:/dev/video0` allows the container to access the host machine's video capture device `/dev/video0`. `--device=/dev/video0` will share the first video device with the container.
- `-it openpose_vin:final /bin/bash` starts a Bash shell in the container for interactive use.

After locally obtaining the Docker image, you have the capability to instantiate multiple instances of this image.

The most frequently utilized commands for managing Docker containers include:

- `docker ps -a`: This command provides a comprehensive list of all Docker containers, including both running and stopped containers, along with relevant details.
- `docker ps`: Executing this command gives you an overview of the currently running Docker containers, displaying key information about the status of each container.
- `docker start container_name`: Use this command to initiate a stopped Docker container by specifying its name, making it operational.
- `docker attach container_name`: This command allows you to attach your terminal session to a specific running Docker container, facilitating interactive interaction with its processes and content.

In this context, the term `container_name` refers to the specific name assigned to an instance of a Docker image, which is essentially a running container.

### 5.2.3 ROS - Openpose v1.7.0

Moving forward, our discussions will exclusively focus on the significant modifications made within the ROS wrapper's launch files inside with **Docker image instance** to ensure seamless compatibility with the Openpose library. We will be closely examining a set of crucial launch files in the following sections, delving deeply into their contents and significance.

- In `/openpose_ros/src/ros_openpose/launch` folder:

```
openpose_docker@shell:~/openpose_ros/src/ros_openpose/launch# ls
config_kinect.launch config_kinectxbox.launch config_nodepth.launch core.launch run.launch

config_kinectxbox.launch
core.launch
run.launch
```



## 5 Realization of Solution

The primary launch file is `run.launch` responsible for initiating the integration of the `openpose` library with ROS by launching three additional files sequentially.

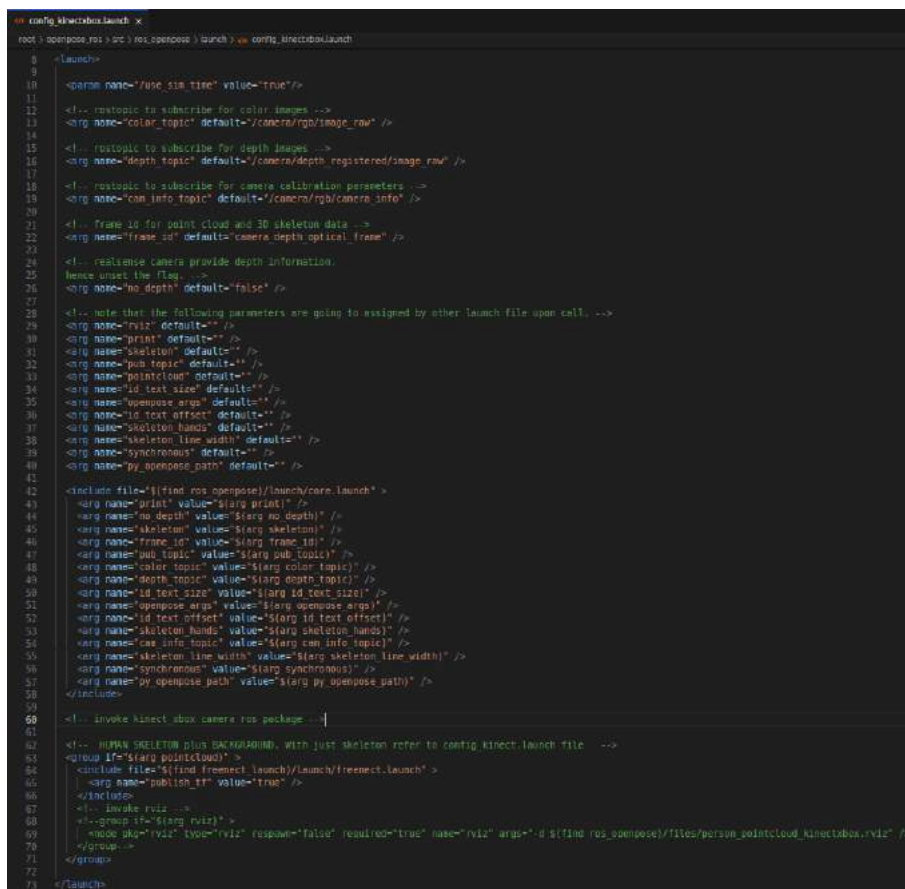
A key parameter of interest within this launch file is a specific command line containing a series of arguments. This command line allows us to customize various aspects of our implementation, including the choice of the model to be utilized (currently set to `BODY_25`), the desired resolution, and whether to include hand and face keypoints or not.

```
<!-- full path to the model dir of openpose -->  
<arg name="openpose_args" value="--net_resolution 128x96 --model_folder /openpose/models/" />
```

Within the `run.launch` file, we initiate the launch of the `config_kinectxbox.launch` file. All the parameters contained within this file hold significant importance. The snippet of the launch file 5.2 includes self-explanatory comments that provide a clear understanding of the purpose and functionality of these parameters.

- ROS command to launch Openpose library

```
openpose_docker@shell:~# roslaunch ros_openpose run.launch
```



```
1  <!-- launch -->  
2  
3  <param name="use_sim_time" value="true" />  
4  
5  <!-- rosnodejs to subscribe for color images -->  
6  <arg name="color_topic" default="/camera/rgb/image_raw" />  
7  
8  <!-- rosnodejs to subscribe for depth images -->  
9  <arg name="depth_topic" default="/camera/depth_registered/image_raw" />  
10  
11 <!-- rosnodejs to subscribe for camera calibration parameters -->  
12 <arg name="cam_info_topic" default="/camera/rgb/camera_info" />  
13  
14 <!-- Frame id for point cloud and 3D skeleton data -->  
15 <arg name="frame_id" default="camera_depth_optical_frame" />  
16  
17 <!-- realSense camera provide depth information, hence unset the flag -->  
18 <arg name="no_depth" default="false" />  
19  
20 <!-- note that the following parameters are going to assigned by other launch file upon call -->  
21 <arg name="rviz" default="" />  
22 <arg name="print" default="" />  
23 <arg name="skeleton" default="" />  
24 <arg name="pub_topic" default="" />  
25 <arg name="pointcloud" default="" />  
26 <arg name="id_text_size" default="" />  
27 <arg name="openpose_args" default="" />  
28 <arg name="id_text_offset" default="" />  
29 <arg name="skeleton_hands" default="" />  
30 <arg name="skeleton_line_width" default="" />  
31 <arg name="synchronous" default="" />  
32 <arg name="py_openpose_path" default="" />  
33  
34 <include file="$(find ros_openpose)/launch/core.launch" >  
35   <arg name="urint" value="$(arg print)" />  
36   <arg name="no_depth" value="$(arg no_depth)" />  
37   <arg name="skeleton" value="$(arg skeleton)" />  
38   <arg name="frame_id" value="$(arg frame_id)" />  
39   <arg name="pub_topic" value="$(arg pub_topic)" />  
40   <arg name="color_topic" value="$(arg color_topic)" />  
41   <arg name="depth_topic" value="$(arg depth_topic)" />  
42   <arg name="id_text_size" value="$(arg id_text_size)" />  
43   <arg name="id_text_offset" value="$(arg id_text_offset)" />  
44   <arg name="openpose_args" value="$(arg openpose_args)" />  
45   <arg name="skeleton_hands" value="$(arg skeleton_hands)" />  
46   <arg name="skeleton_line_width" value="$(arg skeleton_line_width)" />  
47   <arg name="synchronous" value="$(arg synchronous)" />  
48   <arg name="py_openpose_path" value="$(arg py_openpose_path)" />  
49 </include>  
50  
51 <!-- invoke kinect xbox camera ros package -->  
52  
53 <!-- HUPAN SKELETON plus BACKBAND, with just skeleton refer to config_kinect.launch file -->  
54 <group if="$(arg pointcloud)" >  
55   <include file="$(find freenect_launch)/launch/freenect.launch" >  
56     <arg name="publish_tf" value="true" />  
57   </include>  
58  
59   <!-- invoke rviz -->  
60   <!-- group if="$(arg rviz)" >  
61     <node pkg="rviz" type="rviz" respawn="false" required="true" name="rviz" args="$(find ros_openpose)/files/psrsm_pointcloud_kinectxbox_rviz" />  
62   </group -->  
63 </group -->  
64 </launch>
```

Figure 5.2: `config_kinectxbox_launch` file



Within the `config_kinectxbox.launch` file, we trigger the launch of the Camera ROS driver `Freenect` and the `core.launch` file. Our intention is to exclude any `RVIZ` visualization, so we've simply commented out the `RVIZ` visualizer node in the `core.launch` file.

The ROS topic of utmost significance in our context is the `/frame` ROS topic. Our entire research effort revolves around this particular topic, as it furnishes all the requisite information for Human Gesture and Posture estimation.

**Table 5.1:** Frame.msg ROS Custom Message Structure

Field	Description
header	Standard ROS message header
persons[]	Array of persons
bodyParts[]	Array of body parts
leftHandParts[]	Array of left hand parts
rightHandParts[]	Array of right hand parts
faceParts[]	Array of face parts
score	32-bit floating-point score
pixel	Pixel information
point	2D point coordinates
x	32-bit floating-point x-coordinate
y	32-bit floating-point y-coordinate

We have defined two **ROS nodes**. The first one, named `gesture_ctrl.py` (Gestures-based Control Algorithm 5.1), is responsible for controlling the robot's movements through gestures. The second one, `posture_detection.py` (Posture and Facing Detection Algorithm 5.2), is dedicated to estimating human posture. Notably, we have merged the functionalities of these two nodes into a single node, which plays a central role in our Gesture Control Navigation Stack. This unified node is identified as `set_goal.py`.

During the course of our implementation, we encountered certain issues that were subsequently raised on GitHub. You can gain valuable insights into these issues by following this link [REQUIRED process \[rosOpenpose-2\] has died! #88](#).

## 5.3 Communication with Butler

Establishing communication with the Butler Bot is crucial for controlling the robot using gestures and effectively utilizing the Navigation stack. Several essential ROS topics are associated with the robot, including `/tb_cmd_vel`, `/tf`, `/tf_static`, and `/tb_control/wheel_odom`. These topics play a pivotal role in enabling seamless interaction and control of the robot's movements and sensor data.

Initially, access the Ohmni Telepresence robot's shell through the Ohmni app's settings button, found on the blue screen GUI of the robot's interactive screen. Subsequently, perform a sequence of seven taps on the "Version Name" item within the settings menu. This action unveils previously concealed features and options. At the bottom of the menu, locate the "Enable Android Debug Bridge (ADB)"

---

**Algorithm 5.1** Gestures-based Control Algorithm

---

**ROS Node :** gesture\_ctrl.py

**Input data :** 3D and 2D Human Pose

**Output data :** Butler Robot Actions

```
while (obey_me) {
  if (Full Attention) {
    if (Target Person Selection Depending upon Distance) {
      if (Gesture Recognition) {
        if (go_forward_gesture(person)) {
          // Move robot forward
          Execute "GO Forward!"
          robot_movement.forward_movement()
        } else if (disobey_gesture(person)) {
          // Stop obeying
          Execute "DontObey"
          obey_me = False
        } else if (go_right_gesture(person)) {
          // Turn right
          Execute "GO Right!"
          robot_movement.turn_right()
        } else if (go_left_gesture(person)) {
          // Turn left
          Execute "GO Left"
          robot_movement.turn_left()
        } else if (go_backward_gesture(person)) {
          // Move backward
          Execute "GO Backward"
          robot_movement.backward_movement()
        } else if (go_forward_with_right_turn(person)) {
          // Move forward with right turn
          Execute "forward with right turn"
          robot_movement.forward_right_turn()
        } else if (go_forward_with_left_turn(person)) {
          // Move forward with left turn
          Execute "forward with left turn"
          robot_movement.forward_left_turn()
        } else {
          // No recognized gesture
          Execute "NoGesture"
          robot_movement.stop()
        }
      }
    }
  }
}
```

---

**Algorithm 5.2** Posture and Facing Detection Algorithm

---

**ROS Node : posture\_detection.py****Input data : 2D Human Pose****Output data : Publish posture and facing messages, increment person counter.**

1. Check if whole person is in camera frame
  2. If the condition in step 1 is met:
    - Check if the person's posture is facing away using `posture.facing_away(person)`.
      - If true:
        - Log "Person is facing away!"
        - Set `pose_face_msg.facing` to 'FacingSideways'.
      - If false, proceed to the next step.
    - Check if the person's posture is facing back using `posture.facing_back(person)`.
      - If true:
        - Log "Person is facing backward!"
        - Set `pose_face_msg.facing` to 'FacingBackwards'.
      - If false, proceed to the next step.
    - If neither of the above conditions is met:
      - Log "Person is facing towards!"
      - Set `pose_face_msg.facing` to 'FacingTowards'.
  3. Check if the person's posture is sitting using `posture.sitting(person)`.
    - If true:
      - Log "Person is sitting!"
      - Set `pose_face_msg.pose` to 'Sitting'.
    - If false:
      - Log "Person is standing!"
      - Set `pose_face_msg.pose` to 'Standing'.
  4. If the condition in step 1 is not met:
    - Log "Whole person is not in the frame!"
    - Set `pose_face_msg.pose` to 'NoPose'.
    - Set `pose_face_msg.facing` to 'NoFace'.
  5. Publish the `pose_face_msg`.
  6. Increment `person_counter`.
-

option. Activating this option initiates the ADB daemon, granting the capability to establish a connection from the local network. This crucial process enables seamless communication with the robot, enhancing control and interaction.

On your local system, you should execute the following commands [[Detailed Explanation](#)] [37]:

```
IP address localhost : 129.69.207.127
IP address ButlerBot  : 129.69.207.64

localhost@shell:~$ adb connect 129.69.207.64
localhost@shell:~$ adb shell

ButlerBot@shell:~$ su
ButlerBot@shell:~$ setprop ctl.stop tb-node

//Start the Docker Container
ButlerBot@shell:~$ docker start vibrant_chaum; docker attach vibrant_chaum
ButlerBot_docker@shell:~#
```

To connect multiple devices to the same ROS Master, please adhere to the following steps:

- Set the ROS\_IP environment variable to specify the machine's IP address in the ROS network.

```
localhost@shell:~$ export ROS_IP=129.69.207.127
```

- Set the ROS\_MASTER\_URI environment variable to specify the URI of the ROS Master.

```
localhost@shell:~$ export ROS_MASTER_URI=http://129.69.207.64:11311
```

Explanation :

- ROS\_IP is used to define the machine's IP address in the ROS network.
- ROS\_MASTER\_URI specifies the URI of the ROS Master, which manages ROS communication.

## 5.4 Gesture Navigation - Butler

### 5.4.1 Efficient Methodical Approach to Environment Mapping

Mapping the environment is a meticulous and patient process, necessitating a deliberate pace when maneuvering the robot through its surroundings. Sudden changes in orientation or rapid robot movements can introduce errors in the map formation. Additionally, it's crucial to exercise caution when encountering transparent glass surfaces, as they can distort the map and contribute to inaccuracies in the mapping process.

To initiate the mapping of the environment, we commence by launching the tutorial launch file. The following launch file snippet 5.3 provides a concise overview of this procedure.

- ROS command to launch Hector SLAM

```

tutorial.launch M X
src > hector_slam > hector_slam_launch > launch > tutorial.launch
1  <?xml version="1.0"?>
2
3  <launch>
4
5  <include file="$(find ydlidar_ros_driver)/launch/ydlidar.launch"/>
6
7  <arg name="geotiff_map_file_path" default="$(find hector_geotiff)/maps"/>
8
9  <param name="use_sim_time" value="false"/> <!--in real time make it false otherwise when used in simulation make it true-->
10
11 <node pkg="rviz" type="rviz" name="rviz"
12   args="-d $(find hector_slam_launch)/rviz_cfg/mapping_demo.rviz"/>
13
14 <include file="$(find hector_mapping)/launch/mapping_default.launch"/>
15
16 <include file="$(find hector_geotiff_launch)/launch/geotiff_mapper.launch">
17   <arg name="trajectory_source_frame_name" value="scanmatcher_frame"/>
18   <arg name="map_file_path" value="$(arg geotiff_map_file_path)"/>
19 </include>
20
21 </launch>

```

**Figure 5.3:** Hector SLAM launch file

```
localhost@shell:~$ roslaunch hector_slam_launch tutorial.launch
```

The significance of this launch file lies in its pivotal role in configuring and initiating critical components within a robotic system operating on the ROS. Its functionality commences by incorporating configuration files for the YDLidar sensor driver while also establishing default paths for geotiff maps. Furthermore, it defines a ROS parameter to govern the utilization of simulated time by the system.

The launch file proceeds to initiate the ROS Visualization (RVIZ) tool, employing a specialized configuration tailored for the visualization of mapping data. Additionally, it encompasses configuration files for Hector Mapping, a widely utilized ROS package for SLAM tasks. It also incorporates configuration settings for geotiff mapping, which is instrumental in the creation of georeferenced maps. Collectively, these amalgamated configurations build's a comprehensive ROS environment that encompasses mapping, visualization, and map generation functionalities. This environment plays an indispensable role in ensuring the smooth operation of the robotic system.

Once the environment mapping process is complete, preserving the map can be accomplished using the `map_saver` tool, which is part of the `map_server` package. Importantly, this operation should be executed without the need to terminate the Hector SLAM node. When this action is performed, the map is saved under a specified name, generating both a `name_of_the_map.pgm` file and a corresponding `name_of_the_map.yaml` file.

- ROS command to save Environment Map

```
localhost@shell:~$ rosrn map_server map_saver -f name_of_the_map
```

The Portable Gray Map (PGM) file functions as a repository for the map image itself, while the Yet Another Markup Language (YAML) file serves as a repository for crucial map metadata. This combined approach ensures the seamless preservation and management of mapping data without interrupting the ongoing Hector SLAM node operation.

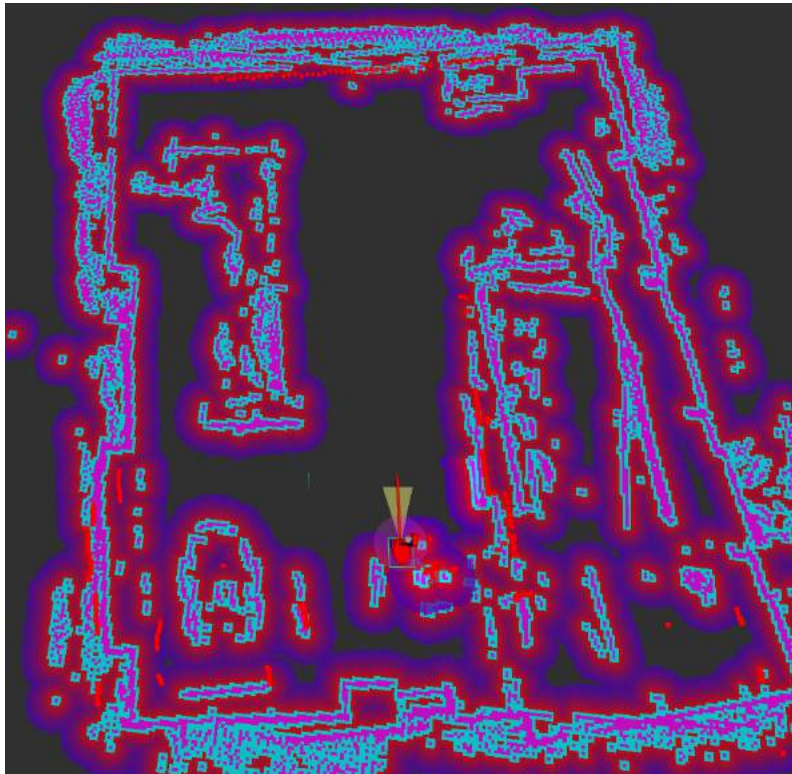
### 5.4.2 Effective Navigation Stack Implementation

By executing the command `rqt_reconfigure`, you can access the Graphical User Interface (GUI) known as `rqt_reconfigure`. This interface facilitates the interactive adjustment of parameters associated with your ROS nodes. This capability proves exceptionally valuable when it comes to refining the performance of your robotic systems or other ROS-dependent applications, all without the need to halt and reboot them.

```
localhost@shell:~$ rosrn rqt_reconfigure rqt_reconfigure
```

- ROS command to launch Navigation stack and AMCL package

```
localhost@shell:~$ roslaunch ohmni_2dnav ohmni_nav.launch
```



**Figure 5.4:** Navigation Map Visualization

Initially, we established the robot's initial position on the map using the RVIZ 2D pose estimate feature. Subsequently, we provided the initial pose via the AMCL package, as previously discussed. To gain insight into the robot's self-perception within its environment, we executed the command `roslaunch tf_echo /map /base_link`. This command activates the `tf_echo` node from the `tf` package, allowing us to observe the transformation data, including translation and rotation, between the `/map` frame and the `/base_link` frame within the robot's `tf` tree. This information plays a crucial role in debugging and ensuring the robot's accurate understanding of its position and orientation in the world, which is fundamental to our task.

```
localhost@shell:~$ rosrn tf tf_echo /map /base_link
At time 1691404176.315
- Translation: [-0.173, 0.630, 0.000]
- Rotation: in Quaternion [0.000, 0.000, 0.997, -0.080]
           in RPY (radian) [0.000, -0.000, -2.982]
           in RPY (degree) [0.000, -0.000, -170.845]
```

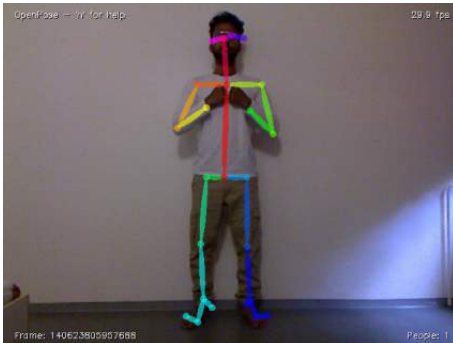
The robot's destination can be specified through two methods: using the RVIZ feature of 2D Nav Goal or by utilizing the ROS node `2d_point_nav.py`.

- ROS command to run `2d_point_nav.py` node

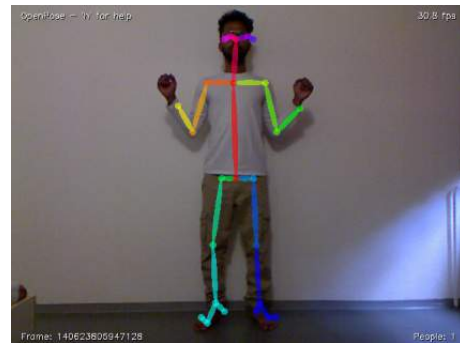
```
localhost@shell:~$ rosrn ohmni_2dnav 2d_point_nav.py
```

#### 5.4.2.1 Gestures-based Navigation

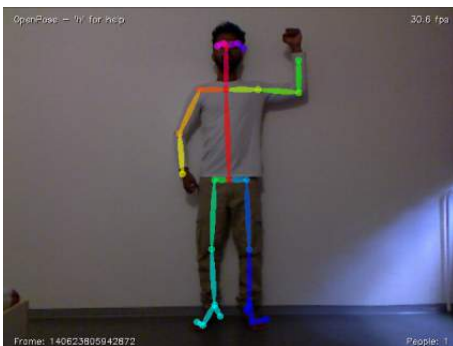
Some modifications have been made to the gesture definitions as depicted in the accompanying Figures.



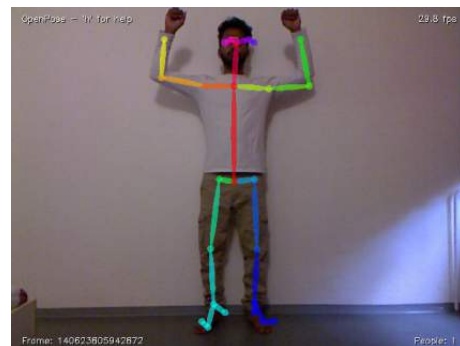
**Figure 5.5:** Obey Gesture



**Figure 5.6:** Disobey Gesture



**Figure 5.7:** Go to Location A



**Figure 5.8:** Go to Location B

To initiate gesture-based navigation, you need to execute the ROS node named `gesture_goal_nav.py` (Gestures-based Navigation Algorithm 5.3).

- ROS command to run `gesture_goal_nav.py` node

```
localhost@shell:~$ rosrn ohmni_2dnav gesture_goal_nav.py
```

Gestures play a pivotal role in the realm of non-verbal HRI and Human-Robot Collaboration (HRC). These non-verbal cues carry substantial importance in facilitating effective communication between human and Butler robot. One particularly noteworthy application of gesture-based commands within this context is their utilization in guiding and instructing Butler robot to achieve navigation objectives. Upon receiving these gesture-based commands, the Butler robot leverages its autonomous capabilities to carry out a series of complex tasks.

- **Goal Interpretation:** The robot's onboard software first interprets the received gestures, translating it into actionable task.
- **Path Planning:** Once the robot comprehends the user's goal via gesture, it proceeds to plan a suitable path to reach the designated destination.
- **Obstacle Avoidance:** During the execution of the planned path, the robot continually scans its environment for dynamic obstacles or unexpected changes. It adjusts its trajectory in real-time to avoid collisions or disruptions, prioritizing safety and adaptability.
- **Autonomous Navigation:** The robot relies on its autonomous navigation system to traverse the environment, making course adjustments as needed while adhering to the prescribed path.
- **Goal Achievement:** Ultimately, the robot's autonomous movements lead it to the predefined destination, successfully completing the assigned task.

In essence, the integration of gesture-based commands into HRI and HRC scenarios empowers Butler to understand and respond to human intentions with remarkable precision. By using gestures to convey goals and instructions, humans can harness the full potential of robots while simplifying the communication process and reducing the need for explicit verbal commands.



---

**Algorithm 5.3** Gestures-based Navigation Algorithm

---

**ROS Node : gesture\_goal\_nav.py****Input data : Human Gestures****Output data : Butler Robot Actions**

1. Initialize global variables:
  - LocationA True
  - LocationB False
2. Check conditions:
  - a. Is ``obeyHuman`` True?
  - b. Is ``person_facing`` 'FacingTowards'?
  - c. Is ``person_pose`` 'Standing'?
3. If all conditions in step 2 are met:
  - a. Check if ``person_gesture`` is 'DeskA' and ``LocationA`` is True:
    - i. If both conditions are met:
      - Move the robot to location (A\_x, A\_y, A\_yaw)
      - Log: "Goal given by person personID"
      - Check if goal reached (result == True):
        - If it was not reached:
          - Log error: "Error: Goal not reached, going back to initial position"
          - Send robot back to initial position (init\_x, init\_y, init\_yaw)
        - If it was reached successfully:
          - Log: "Goal A reached successfully"
          - Update global variables:
            - Set LocationA False
            - Set LocationB True
  - b. Check if ``person_gesture`` is 'DeskB' and ``LocationB`` is True:
    - i. If both conditions are met:
      - Move the robot to location (B\_x, B\_y, B\_yaw)
      - Log: "Goal given by person personID"
      - Check if goal reached (result == True):
        - If it was not reached:
          - Log error: "Error: Goal not reached, going back to initial position"
          - Send robot back to initial position (init\_x, init\_y, init\_yaw)
        - If it was reached successfully:
          - Log: "Goal B reached successfully"
          - Update global variables:
            - Set LocationA True
            - Set LocationB False

## 5.5 Time synchronization on Multiple Machine's

Initially, there was a time difference of approximately 2 hours between the Robot Machine and the Local System. To address this, both systems were adjusted to be in the same time zone. However, despite this synchronization, a persistent time asynchrony issue remained, resulting in a delay of about 3-4 seconds. This delay had significant implications, leading to two critical problems:

- The timely update of Lidar Scan data within the Map environment was hindered. Consequently, the entire navigation of the robot and the Navigation stack were adversely affected.
- The SLAM algorithm interpreted the incoming data from the Robot machine as outdated TF (Transform) data. This exacerbation of the situation created further complications in the robot's navigation [Error: TF\\_OLD\\_DATA](#).

To resolve the issue of time asynchrony, we leveraged the assistance of the **chrony** tool. Here is some pertinent information about **Chrony** [10]:

**Chrony** is a highly versatile implementation of the Network Time Protocol (NTP), renowned for its ability to synchronize system clocks with precision. It accomplishes this synchronization through various means, including NTP servers, reference clocks such as GPS receivers, and manual inputs facilitated by both wristwatches and keyboard commands.

In terms of time accuracy, Chrony consistently delivers impressive results. When synchronizing two machines over the Internet, it typically achieves synchronization within just a few milliseconds, demonstrating its reliability for remote timekeeping. In the context of a Local Area Network (LAN), Chrony attains accuracy levels typically in the range of tens of microseconds, ensuring precision for internal network synchronization. For even higher levels of accuracy, the use of hardware timestamping or a hardware reference clock with Chrony may make sub-microsecond precision feasible.

In a server-client configuration, the client system is configured to adhere to the time settings of the server, ensuring synchronized timekeeping across both machines. To achieve this synchronization between the local system and the robot machine, specific adjustments were made within the `'/etc/chrony/chrony.conf'` file on both devices.

- **Chrony Config file on Host machine : Server**

```
#by default its acting as cilent so we have to make it Server
local stratum 8

# allowed clients
allow 129.69.0.0/16

# This directive specify the location of the file containing ID/key pairs for
# NTP authentication.
keyfile /etc/chrony/chrony.keys

# This directive specify the file into which chronyd will store the rate
# information.
driftfile /var/lib/chrony/chrony.drift
```

```
# Log files location.
logdir /var/log/chrony

# Stop bad estimates upsetting machine clock.
maxupdateskew 100.0

# This directive enables kernel synchronisation (every 11 minutes) of the
# real-time clock. Note that it can't be used along with the 'rtcfile' directive.
rtcsync

# Step the system clock instead of slewing it if the adjustment is larger than
# one second, but only in the first three clock updates.
makestep 1 3
```

- **Chrony Config file on Butler machine : CLIENT**

```
server 129.69.207.127 ibur
# This directive specify the location of the file containing ID/key pairs for
# NTP authentication.
keyfile /etc/chrony/chrony.keys

# This directive specify the file into which chronyd will store the rate
# information.
driftfile /var/lib/chrony/chrony.drift

# Uncomment the following line to turn logging on.
#log tracking measurements statistics

# Log files location.
logdir /var/log/chrony

# Stop bad estimates upsetting machine clock.
maxupdateskew 100.0

# This directive enables kernel synchronisation (every 11 minutes) of the
# real-time clock. Note that it can't be used along with the 'rtcfile' directive.
rtcsync

#Step the system clock instead of slewing it if the adjustment is larger than
# one second, but only in the first three clock updates.
makestep 1 3
```

Following the modifications to the file, the respective machines' terminals were utilized to execute the following commands:

## 5 Realization of Solution

---

```
# commands on local system
localhost@shell:~$ sudo systemctl restart chrony
localhost@shell:~$ sudo systemctl status chrony
localhost@shell:~$ sudo systemctl enable chrony
```

```
# commands on robot machine
ButlerBot_docker@shell:~# sudo /etc/init.d/chrony restart
ButlerBot_docker@shell:~# sudo systemctl enable chrony
```

These are crucial links that were referenced to address the time asynchrony issue:

- [ROS Timestamps](#)
- [Tf timeout with multiple machines](#)
- [Use chrony to synchronize timestamp on two machines](#)

## 6 Evaluation of Solution

### 6.1 Assessing Posture and Gesture Accuracy for Robot Control

Gesture accuracy is assessed through a series of tests involving the execution of 8 gestures, and the subsequent measurement of their success rates. These tests were carried out with a human positioned approximately 2 meters in front of a robot equipped with a Kinect camera. Each specific gesture and posture was meticulously performed and repeated 50 times, totaling 50 trials for each gesture and posture.

Gesture	Number of times identified	Accuracy (%)
obey_gesture	50	100%
disobey_gesture	50	100%
go_forward_gesture	50	100%
go_backward_gesture	50	100%
go_right_gesture	46	92%
go_left_gesture	45	90%
go_forward_with_right_turn	44	88%
go_forward_with_left_turn	45	90%

**Table 6.1:** Gesture Detection Results

The precision of postures is assessed through the execution of specific postures and the subsequent measurement of their success rates. These evaluations are conducted with a human positioned at a variable distance, typically ranging from 2 to 6 meters, in front of a robot equipped with a Kinect camera.

Posture	Number of times identified	Accuracy (%)
Facing Towards	50	100%
Facing Backwards	50	100%
Facing Sideways	50	100%
Sitting (Facing Towards Robot)	46	92%
Sitting (Facing Sideways w.r.t Robot)	50	100%
Standing	48	96%

**Table 6.2:** Posture Detection Results

The pose estimation was accomplished using the Openpose library, facilitated by a ROS wrapper, operating at an impressive frame rate of nearly 30 frames per second and at a resolution of 160x160 pixels. The primary objective of these tests was to evaluate the feasibility of controlling the robot's movements through gesture recognition.

## 6.2 Assessing Posture and Gesture Accuracy for Robot Navigation

The accuracy of gesture-based robot navigation remains consistently at 100% for the following gestures: Obey Gesture, Disobey Gesture, Go to Location A, and Go to Location B. These accuracy assessments were conducted through a series of tests involving the execution of these four gestures, with a focus on evaluating their success rate. During these tests, the distance between the human operator and the robot was approximately 4 meters.

For the purpose of pose estimation, we employed the Openpose library, integrated within a ROS wrapper. This system operated at a frame rate of approximately 21 frames per second, utilizing a resolution of 128x96 pixels. The primary objective of these tests was to assess the feasibility of controlling the robot's autonomous navigation through gesture recognition.

To ensure robust testing, each specific gesture was performed and repeated 20 times, resulting in a total of 20 trials for each gesture and corresponding posture. It's worth noting that we had to lower the resolution during these experiments, as the frame rate experienced a significant drop when both the Navigation Stack and Localization package were running simultaneously.

In the course of conducting 20 trials, we computed the discrepancies between the actual and desired coordinates (both in terms of x and y) as well as the robot's orientation. Notably, the initial location and Location A were separated by a distance of 3 meters, while the initial position and Location B were separated by a distance of 6 meters.

Robot's Movement	Average Percentage Error for (x, y) Coordinates	Average Percentage Error for orientation $\theta$
Initial location to Location A	4.4%	4.1%
Initial location to Location B	9.7%	6.9%

**Table 6.3:** Robot Position Deviation and Error Percentages

## 6.3 Enhancing Robot Performance: Balancing GPU Utilization and Computational Overheads

- Our diligent efforts have been focused on preserving our hardware's longevity during prolonged usage, while also ensuring the **real-time** capability of our implementation by consistently maintaining Graphics Processing Unit (GPU) utilization within the 80-90% range. When both the Openpose and Navigation Stack are running simultaneously, Central Processing Unit (CPU) utilization rarely exceeds 60%. Notably, the Navigation Stack minimally utilizes the GPU for its execution. To monitor GPU utilization, we have implemented a check by adjusting the image resolution provided to the Openpose library, with a compromise on accuracy.

### 6.3 Enhancing Robot Performance: Balancing GPU Utilization and Computational Overheads

---

Decreasing the resolution adversely impacts the accuracy of the Openpose algorithm. The highest workable resolution with these algorithms is 256x256; beyond this, the frame rate drops to 6-8 FPS, and GPU utilization spikes up to 93%.

- Incorporating hand keypoint detection into the Openpose algorithm introduces significant computational overhead, resulting in a reduced FPS of 7 with a 256x256 resolution and approximately 91% GPU utilization. This leads to the introduction of an additional 42 keypoints, bringing the total number of keypoints to 67 (25 keypoints of Body part plus 42 keypoints of Hands). Consequently, this results in an elevation of computational complexity. Consequently, this configuration has seen limited use in our research work [[Hand Output Format](#)] [18].
- We've witnessed impressive improvements in the robot's response to dynamic obstacles following necessary adjustments to the navigation stack.
- Additionally, it's worth noting two crucial considerations:
  - The strength and reliability of the local network, and
  - Ensuring the robot is fully charged, or at least above 80%, to ensure smooth operation.





# 7 Conclusion

## 7.1 Summary

The research study explores Gesture and Posture recognition as a means of facilitating communication and control between humans and Butler robot. This involves the translation of human poses into command signals, actions to trigger state changes, and initiation of information exchange with the robot. Utilizing a custom Docker image for Openpose, a successful implementation of gesture-based control for the Butler Robot's movement, coupled with the accurate determination of human posture within the frame, has been achieved. To prioritize safety, crucial social rules have been embedded to ensure secure interactions between humans and robots.

The robot captures visual data through its camera system, enabling informed decisions to be made, as well as the identification and classification of human movements. Activity recognition, aimed at understanding human movements, empowers robots to anticipate and identify human actions within their environment. This, in turn, enables the robot to offer valuable information, guidance, or assistance across various contexts.

An adaptation of the Hector SLAM algorithm was necessary due to changes in the office environment, resulting in impressive map creation results. The navigation stack setup was meticulously redefined, making subtle yet significant adjustments that greatly impacted the robot's navigation. The simplicity and organized structure of the navigation stack played a pivotal role in achieving these remarkable outcomes. Despite a limited time-frame for improvement, the results achieved in the autonomous navigation of the Butler Robot are truly impressive compared to previous work [2] and [3].

The successful implementation of gesture-based navigation for the robot involved integrating Openpose's custom Docker image with the Navigation Stack. Notably, the Navigation Stack and Localization algorithm utilize the CPU, while the Openpose algorithm significantly leverages the GPU. A direct relationship between frame resolution and the accuracy of human keypoint detection is evident. Higher frame resolution enhances accuracy but demands greater computational power, resulting in reduced Frames Per Second (FPS) and hindering real-time functionality. Striking a well-balanced compromise between frame resolution, accuracy, and FPS allowed the objectives of the thesis to be successfully realized, creating a real-time implementation.

## 7.2 Future Work

The thesis work presents several potential avenues for improvement and extension, which are outlined as follows:

- **Enhancing Camera Placement:** One avenue for improvement involves optimizing the Microsoft Kinect RGB-D Camera's placement on the robot and implementing the Real-Time Appearance-Based Mapping (RTAB-Map), an RGB-D Graph SLAM approach. This enhancement would empower the robot to detect obstacles in 3D space, thereby enhancing the system's overall efficiency.
- **Human Tracking or Sorting:** Another promising direction for further development is the implementation of a sorting technique or human tracking system. This addition would enable the robot to selectively respond to specific human commands while disregarding others present in the image frame, enhancing the robot's responsiveness in dynamic environments.
- **Dynamic Gesture Recognition:** To expand the scope of the research, there is the opportunity to transition from static gestures to dynamic gestures for controlling robot actions. This extension would involve defining a wider range of gestures and improving the logic, particularly for recognizing sitting postures, thereby increasing the versatility and adaptability of the gesture control system.

## Bibliography

- [1] N. Aboki, I. Georgievski, M. Aiello. “Automating a Telepresence Robot for Human Detection, Tracking, and Following”. en. In: *Towards Autonomous Robotic Systems*. Ed. by F. Iida, P. Maiolino, A. Abdulali, M. Wang. Vol. 14136. Series Title: Lecture Notes in Computer Science. Cham: Springer Nature Switzerland, 2023, pp. 150–161. ISBN: 978-3-031-43359-7 978-3-031-43360-3. DOI: [10.1007/978-3-031-43360-3\\_13](https://doi.org/10.1007/978-3-031-43360-3_13). URL: [https://link.springer.com/10.1007/978-3-031-43360-3\\_13](https://link.springer.com/10.1007/978-3-031-43360-3_13) (visited on 09/28/2023) (cit. on p. 60).
- [2] E. Arfa. “Study and implementation of LiDAR-based SLAM algorithm and map-based autonomous navigation for a telepresence robot to be used as a chaperon for smart laboratory requirements”. en. MA thesis (cit. on pp. 32, 33, 57, 89).
- [3] S. P. Avinash Kangare. *Multi-Application Implementation Using Path Planning and Computer Vision*. en. Tech. rep. Institute of Architecture of Application Systems, 2022 (cit. on pp. 57, 60, 89).
- [4] J.-H. Baek, J.-H. Choi, S.-M. Kim, H.-J. Park, T.-Y. Kuc. “A Mobile Robot Framework in Industrial Disaster for Human Rescue”. en. In: *2022 22nd International Conference on Control, Automation and Systems (ICCAS)*. Jeju, Korea, Republic of: IEEE, Nov. 2022, pp. 623–628. ISBN: 978-89-93215-24-3. DOI: [10.23919/ICCAS55662.2022.10003936](https://doi.org/10.23919/ICCAS55662.2022.10003936). URL: <https://ieeexplore.ieee.org/document/10003936/> (visited on 08/18/2023) (cit. on p. 60).
- [5] A. Bellarbi, S. Kahlouche, N. Achour, N. Ouadah. “A social planning and navigation for tour-guide robot in human environment”. en. In: *2016 8th International Conference on Modelling, Identification and Control (ICMIC)*. Algiers, Algeria: IEEE, Nov. 2016, pp. 622–627. ISBN: 978-0-9567157-7-7. DOI: [10.1109/ICMIC.2016.7804186](https://doi.org/10.1109/ICMIC.2016.7804186). URL: <http://ieeexplore.ieee.org/document/7804186/> (visited on 08/18/2023) (cit. on p. 36).
- [6] *Camera Calibration*. URL: <https://wiki.ros.org/roslaunch/Tutorials/IntrinsicCalibration> (cit. on p. 68).
- [7] Z. Cao, G. Hidalgo Martinez, T. Simon, S. Wei, Y. A. Sheikh. “OpenPose: Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2019) (cit. on pp. 25–30, 68).
- [8] Z. Cao, G. Hidalgo, T. Simon, S.-E. Wei, Y. Sheikh. *OpenPose: Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields*. en. arXiv:1812.08008 [cs]. May 2019. URL: <http://arxiv.org/abs/1812.08008> (visited on 08/25/2023) (cit. on p. 35).
- [9] Z. Cao, T. Simon, S.-E. Wei, Y. Sheikh. “Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields”. In: *CVPR*. 2017 (cit. on p. 24).
- [10] *Chrony*. URL: <https://chrony-project.org/> (cit. on p. 82).
- [11] A. Cosgun, H. Christensen. *Context Aware Robot Navigation using Interactively Built Semantic Maps*. en. arXiv:1710.08682 [cs]. Aug. 2018. URL: <http://arxiv.org/abs/1710.08682> (visited on 08/22/2023).

- [12] F. De Assis Moura Pimentel, P. T. Aquino. “Proposal of a New Model for Social Navigation Based on Extraction of Social Contexts from Ontology in Service Robots”. en. In: *2019 Latin American Robotics Symposium (LARS), 2019 Brazilian Symposium on Robotics (SBR) and 2019 Workshop on Robotics in Education (WRE)*. Rio Grande, Brazil: IEEE, Oct. 2019, pp. 144–149. ISBN: 978-1-72814-268-5. DOI: 10.1109/LARS-SBR-WRE48964.2019.00033. URL: <https://ieeexplore.ieee.org/document/9018600/> (visited on 08/21/2023) (cit. on p. 36).
- [13] *Develop Robotics application with ROS*. URL: [https://docs.ohmnilabs.com/develop\\_with\\_ros/](https://docs.ohmnilabs.com/develop_with_ros/) (cit. on p. 24).
- [14] *Docker Introduction*. URL: <https://www.geeksforgeeks.org/introduction-to-docker/> (cit. on p. 23).
- [15] *Docker Openpose*. URL: <https://janbkk10.medium.com/build-to-openpose-docker-on-ssh-server-5603874834e9> (cit. on p. 68).
- [16] G. Doisy, A. Jevtic, S. Bodirosa. “Spatially unconstrained, gesture-based human-robot interaction”. en. In: *2013 8th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*. Tokyo, Japan: IEEE, Mar. 2013, pp. 117–118. ISBN: 978-1-4673-3101-2 978-1-4673-3099-2 978-1-4673-3100-5. DOI: 10.1109/HRI.2013.6483529. URL: <http://ieeexplore.ieee.org/document/6483529/> (visited on 08/18/2023).
- [17] M. Filipenko, I. Afanasyev. “Comparison of Various SLAM Systems for Mobile Robot in an Indoor Environment”. en. In: *2018 International Conference on Intelligent Systems (IS)*. Funchal - Madeira, Portugal: IEEE, Sept. 2018, pp. 400–407. ISBN: 978-1-5386-7097-2. DOI: 10.1109/IS.2018.8710464. URL: <https://ieeexplore.ieee.org/document/8710464/> (visited on 08/18/2023) (cit. on p. 57).
- [18] *Hand Output Format*. URL: [https://github.com/CMU-Perceptual-Computing-Lab/openpose/blob/master/doc/02\\_output.md#hand-output-format](https://github.com/CMU-Perceptual-Computing-Lab/openpose/blob/master/doc/02_output.md#hand-output-format) (cit. on p. 87).
- [19] *Hector SLAM*. URL: [https://wiki.ros.org/hector\\_slam](https://wiki.ros.org/hector_slam) (cit. on p. 32).
- [20] Husarion. *HUSARION-ROS Tutorials*. URL: <https://husarion.com/tutorials/> (cit. on p. 61).
- [21] R. P. Joshi, A. Choi, X. Z. Tan, M. K. Van den Broek, R. Luo, B. Choi. *ROS OpenPose*. [https://github.com/ravijo/ros\\_openpose](https://github.com/ravijo/ros_openpose). 2019 (cit. on pp. 36, 44, 51, 68).
- [22] F. Jumel, J. Saraydaryan, R. Leber, L. Matignon, E. Lombardi, C. Wolf, O. Simonin. “Context Aware Robot Architecture, Application to the RoboCup@Home Challenge”. en. In: *RoboCup 2018: Robot World Cup XXII*. Ed. by D. Holz, K. Genter, M. Saad, O. Von Stryk. Vol. 11374. Series Title: Lecture Notes in Computer Science. Cham: Springer International Publishing, 2019, pp. 205–216. ISBN: 978-3-030-27543-3 978-3-030-27544-0. DOI: 10.1007/978-3-030-27544-0\_17. URL: [https://link.springer.com/10.1007/978-3-030-27544-0\\_17](https://link.springer.com/10.1007/978-3-030-27544-0_17) (visited on 08/22/2023) (cit. on p. 35).
- [23] R. Kabir, N. Ahmed, N. Roy, M. R. Islam. “A Novel Dynamic Hand Gesture and Movement Trajectory Recognition model for Non-Touch HRI Interface”. en. In: *2019 IEEE Eurasia Conference on IOT, Communication and Engineering (ECICE)*. Yunlin, Taiwan: IEEE, Oct. 2019, pp. 505–508. ISBN: 978-1-72812-501-5. DOI: 10.1109/ECICE47484.2019.8942691. URL: <https://ieeexplore.ieee.org/document/8942691/> (visited on 08/18/2023).
- [24] *Kinect XBOX 360 Custom Mobile Adapter*. URL: <https://youtu.be/WsYfIaMJBUQ?si=4KblbnhQVQi5TMMm> (cit. on p. 40).

- [25] Z.-X. Li, G.-H. Cui, C.-L. Li, Z.-S. Zhang. “Comparative Study of Slam Algorithms for Mobile Robots in Complex Environment”. en. In: *2021 6th International Conference on Control, Robotics and Cybernetics (CRC)*. Shanghai, China: IEEE, Oct. 2021, pp. 74–79. ISBN: 978-1-66542-437-0. DOI: [10.1109/CRC52766.2021.9620122](https://doi.org/10.1109/CRC52766.2021.9620122). URL: <https://ieeexplore.ieee.org/document/9620122/> (visited on 08/18/2023) (cit. on p. 57).
- [26] X. Li, Z. Chen, Z. Zhong, J. Ma. “Human-machine Collaboration Method Based on Key Nodes of Human Posture”. en. In: *2022 IEEE Asia-Pacific Conference on Image Processing, Electronics and Computers (IPEC)*. Dalian, China: IEEE, Apr. 2022, pp. 140–146. ISBN: 978-1-66540-902-5. DOI: [10.1109/IPEC54454.2022.9777570](https://doi.org/10.1109/IPEC54454.2022.9777570). URL: <https://ieeexplore.ieee.org/document/9777570/> (visited on 08/21/2023).
- [27] J. A. Mahmud, B. C. Das, J. Shin, K. M. Hasib, R. Sadik, M. F. Mridha. “3D Gesture Recognition and Adaptation for Human–Robot Interaction”. en. In: *IEEE Access* 10 (2022), pp. 116485–116513. ISSN: 2169-3536. DOI: [10.1109/ACCESS.2022.3218679](https://doi.org/10.1109/ACCESS.2022.3218679). URL: <https://ieeexplore.ieee.org/document/9933761/> (visited on 08/18/2023) (cit. on p. 36).
- [28] S. Malik. *Lidar SLAM: The Ultimate Guide to Simultaneous Localization and Mapping*. URL: <https://www.wevolver.com/article/lidar-slam-the-ultimate-guide-to-simultaneous-localization-and-mapping> (cit. on pp. 31, 32).
- [29] D. Martinelli, A. L. Sousa, M. E. Augusto, V. C. Kalempa, A. S. D. Oliveira, R. F. Rohrich, M. A. Teixeira. “Remote Control for Mobile Robots Using Gestures Captured by the RGB Camera and Recognized by Deep Learning Techniques”. en. In: *2019 Latin American Robotics Symposium (LARS), 2019 Brazilian Symposium on Robotics (SBR) and 2019 Workshop on Robotics in Education (WRE)*. Rio Grande, Brazil: IEEE, Oct. 2019, pp. 98–103. ISBN: 978-1-72814-268-5. DOI: [10.1109/LARS-SBR-WRE48964.2019.00025](https://doi.org/10.1109/LARS-SBR-WRE48964.2019.00025). URL: <https://ieeexplore.ieee.org/document/9018623/> (visited on 08/18/2023) (cit. on p. 36).
- [30] J. Miller, S. Hong, J. Lu. “Self-Driving Mobile Robots Using Human-Robot Interactions”. en. In: *2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. Miyazaki, Japan: IEEE, Oct. 2018, pp. 1251–1256. ISBN: 978-1-5386-6650-0. DOI: [10.1109/SMC.2018.00219](https://doi.org/10.1109/SMC.2018.00219). URL: <https://ieeexplore.ieee.org/document/8616215/> (visited on 08/21/2023) (cit. on pp. 36, 60).
- [31] A. Moniz, B.-J. Krings. “Robots Working with Humans or Humans Working with Robots? Searching for Social Dimensions in New Human-Robot Interaction in Industry”. en. In: *Societies* 6.3 (Aug. 2016), p. 23. ISSN: 2075-4698. DOI: [10.3390/soc6030023](https://doi.org/10.3390/soc6030023). URL: <http://www.mdpi.com/2075-4698/6/3/23> (visited on 08/22/2023) (cit. on pp. 13, 35).
- [32] “Navigation Goal”. In: (). URL: <https://automaticaddison.com/how-to-send-goals-to-the-ros-navigation-stack-using-c/>.
- [33] *Navigation Stack Setup*. URL: <http://wiki.ros.org/navigation> (cit. on pp. 61, 62).
- [34] Q. H. Nguyen, P. Johnson, D. Latham. “Performance Evaluation of ROS-Based SLAM Algorithms for Handheld Indoor Mapping and Tracking Systems”. en. In: *IEEE Sensors Journal* 23.1 (Jan. 2023), pp. 706–714. ISSN: 1530-437X, 1558-1748, 2379-9153. DOI: [10.1109/JSEN.2022.3224224](https://doi.org/10.1109/JSEN.2022.3224224). URL: <https://ieeexplore.ieee.org/document/9967952/> (visited on 08/18/2023) (cit. on p. 57).
- [35] *NVIDIA Configuring Docker*. URL: <https://docs.nvidia.com/datacenter/cloud-native/container-toolkit/latest/install-guide.html#> (cit. on p. 68).

- [36] *Ohmni Docker - Overview*. URL: <https://docs.ohmnilabs.com/ohmnidocker/> (cit. on p. 24).
- [37] *Ohmni ROS Developer*. URL: [https://docs.ohmnilabs.com/develop\\_with\\_ros/](https://docs.ohmnilabs.com/develop_with_ros/) (cit. on p. 76).
- [38] *ohmni-telepresence-robot*. URL: <https://ohmnilabs.com/products/ohmni-telepresence-robot/> (cit. on p. 13).
- [39] *ohmni-telepresence-robot Developer*. URL: [https://docs.ohmnilabs.com/develop\\_with\\_ros/](https://docs.ohmnilabs.com/develop_with_ros/) (cit. on p. 17).
- [40] V. Patil. *Openpose Docker Image*. Aug. 2023. URL: [https://hub.docker.com/repository/docker/vin8/openpose\\_vin/general](https://hub.docker.com/repository/docker/vin8/openpose_vin/general) (cit. on p. 70).
- [41] V. V. Patil. *Computer Vision in Robotics*. en. Tech. rep.
- [42] P. K. Prasad, W. Ertel. “Knowledge Acquisition and Reasoning Systems for Service Robots: A Short Review of the State of the Art”. en. In: *2020 5th International Conference on Robotics and Automation Engineering (ICRAE)*. Singapore, Singapore: IEEE, Nov. 2020, pp. 36–45. ISBN: 978-1-72818-981-9. DOI: 10.1109/ICRAE50850.2020.9310835. URL: <https://ieeexplore.ieee.org/document/9310835/> (visited on 08/22/2023) (cit. on pp. 15, 35).
- [43] S. Qiao, Y. Wang, J. Li. “Real-time human gesture grading based on OpenPose”. en. In: *2017 10th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI)*. Shanghai: IEEE, Oct. 2017, pp. 1–6. ISBN: 978-1-5386-1937-7. DOI: 10.1109/CISP-BMEI.2017.8301910. URL: <http://ieeexplore.ieee.org/document/8301910/> (visited on 08/18/2023) (cit. on p. 36).
- [44] N. Robinson, B. Tidd, D. Campbell, D. Kulić, P. Corke. “Robotic Vision for Human-Robot Interaction and Collaboration: A Survey and Systematic Review”. en. In: *ACM Transactions on Human-Robot Interaction* 12.1 (Mar. 2023). arXiv:2307.15363 [cs], pp. 1–66. ISSN: 2573-9522, 2573-9522. DOI: 10.1145/3570731. URL: <http://arxiv.org/abs/2307.15363> (visited on 09/20/2023) (cit. on p. 35).
- [45] emanual robotis. *TurtleBot*. URL: <https://emanual.robotis.com/docs/en/platform/turtlebot3/overview/#overview> (cit. on p. 61).
- [46] *RobotSetup Navigation Stack Setup*. URL: <http://wiki.ros.org/navigation/Tutorials/RobotSetup> (cit. on p. 63).
- [47] *ROS AMCL package*. URL: <http://wiki.ros.org/amcl> (cit. on pp. 33, 43, 65).
- [48] *ROS Camera - Freenect Driver*. URL: [https://wiki.ros.org/freenect\\_launch](https://wiki.ros.org/freenect_launch) (cit. on p. 67).
- [49] *ROS Camera - Openni Driver*. URL: [https://wiki.ros.org/openni\\_launch](https://wiki.ros.org/openni_launch) (cit. on p. 67).
- [50] *ROS cheatsheet's*. URL: <https://github.com/ros/cheatsheet/releases> (cit. on p. 22).
- [51] *ROS Cost Map 2D package*. URL: [http://wiki.ros.org/costmap\\_2d](http://wiki.ros.org/costmap_2d) (cit. on p. 43).
- [52] *ROS dwa\_local\_planner package*. URL: [http://wiki.ros.org/dwa\\_local\\_planner](http://wiki.ros.org/dwa_local_planner) (cit. on p. 43).
- [53] *ROS global\_planner package*. URL: [http://wiki.ros.org/global\\_planner](http://wiki.ros.org/global_planner) (cit. on p. 43).
- [54] *ROS Navigation Stack*. URL: <https://automaticaddison.com/how-to-set-up-the-ros-navigation-stack-on-a-robot/> (cit. on p. 61).
- [55] *ROS Wiki*. URL: <https://wiki.ros.org/> (cit. on pp. 20–22).

- [56] *RPLidar Hector SLAM*. URL: [https://github.com/NickL77/RPLidar\\_Hector\\_SLAM/tree/master](https://github.com/NickL77/RPLidar_Hector_SLAM/tree/master) (cit. on p. 58).
- [57] T. Simon, H. Joo, I. Matthews, Y. Sheikh. “Hand Keypoint Detection in Single Images using Multiview Bootstrapping”. In: *CVPR*. 2017.
- [58] K.-T. Song, P.-C. Lu, S.-H. Song. “Human-Robot Interaction Design Based on Specific Person Finding and Localization of a Mobile Robot”. en. In: *2019 International Automatic Control Conference (CACCS)*. Keelung, Taiwan: IEEE, Nov. 2019, pp. 1–6. ISBN: 978-1-72813-846-6. DOI: 10.1109/CACS47674.2019.9024734. URL: <https://ieeexplore.ieee.org/document/9024734/> (visited on 08/18/2023) (cit. on pp. 36, 60).
- [59] P. Tanugraha. *Understanding OpenPose (with code reference)*. URL: <https://medium.com/analytics-vidhya/understanding-openpose-with-code-reference-part-1-b515ba0bbc73>.
- [60] M. Tenney. *Microsoft Kinect – Hardware*. URL: <https://gmvcast.uark.edu/scanning/hardware/microsoft-kinect-resourceshardware/> (cit. on pp. 39, 40).
- [61] *Turtle Bot - Nav parameters*. URL: <https://github.com/ROBOTIS-GIT/turtlebot3> (cit. on p. 61).
- [62] H. Ukida, K. Tanaka. “Mobile robot operation by gesture recognition using continuous human motion”. en. In: *2015 54th Annual Conference of the Society of Instrument and Control Engineers of Japan (SICE)*. Hangzhou, China: IEEE, July 2015, pp. 1–6. ISBN: 978-4-907764-48-7. DOI: 10.1109/SICE.2015.7285358. URL: <http://ieeexplore.ieee.org/document/7285358/> (visited on 08/18/2023) (cit. on p. 36).
- [63] viso.ai. *Pose Estimation: The ultimate overview in 2021*. URL: <https://viso.ai/deep-learning/pose-estimation-ultimate-overview/>.
- [64] S.-E. Wei, V. Ramakrishna, T. Kanade, Y. Sheikh. “Convolutional pose machines”. In: *CVPR*. 2016.
- [65] F. Xia, L. Tyoan, Z. Yang, I. Uzoije, G. Zhang, P. A. Vela. “Human-aware mobile robot exploration and motion planner”. en. In: *SoutheastCon 2015*. Fort Lauderdale, FL, USA: IEEE, Apr. 2015, pp. 1–4. ISBN: 978-1-4673-7300-5. DOI: 10.1109/SECON.2015.7133021. URL: <http://ieeexplore.ieee.org/document/7133021/> (visited on 08/18/2023) (cit. on pp. 36, 60).
- [66] R. Yagfarov, M. Ivanou, I. Afanasyev. “Map Comparison of Lidar-based 2D SLAM Algorithms Using Precise Ground Truth”. en. In: *2018 15th International Conference on Control, Automation, Robotics and Vision (ICARCV)*. Singapore: IEEE, Nov. 2018, pp. 1979–1983. ISBN: 978-1-5386-9582-1. DOI: 10.1109/ICARCV.2018.8581131. URL: <https://ieeexplore.ieee.org/document/8581131/> (visited on 08/18/2023) (cit. on p. 57).
- [67] Y. Yang, H. Yan, M. Dehghan, M. H. Ang. “Real-time human-robot interaction in complex environment using kinect v2 image recognition”. en. In: *2015 IEEE 7th International Conference on Cybernetics and Intelligent Systems (CIS) and IEEE Conference on Robotics, Automation and Mechatronics (RAM)*. Siem Reap, Cambodia: IEEE, July 2015, pp. 112–117. ISBN: 978-1-4673-7337-1 978-1-4673-7338-8. DOI: 10.1109/ICCIS.2015.7274606. URL: <http://ieeexplore.ieee.org/document/7274606/> (visited on 08/21/2023).
- [68] *YD Lidar*. URL: [https://www.ydlidar.com/Public/upload/files/2022-07-07/YDLIDAR%20X4%20Lidar%20User%20Manual%20V1.3\(211230\).pdf](https://www.ydlidar.com/Public/upload/files/2022-07-07/YDLIDAR%20X4%20Lidar%20User%20Manual%20V1.3(211230).pdf) (cit. on p. 40).



- [69] *YD LiDar X4 Development Manual*. URL: <https://www.ydlidar.com/dowfile.html?cid=5&type=3> (cit. on p. 41).
- [70] *YD LiDar X4 User Manual*. URL: <https://www.ydlidar.com/dowfile.html?cid=5&type=2> (cit. on p. 41).
- [71] C.L. Yong, B. Hoe Kwan, D. W.-K. Ng, H. Seng Sim. “Human Tracking and Following using Machine Vision on a Mobile Service Robot”. en. In: *2022 IEEE 10th Conference on Systems, Process & Control (ICSPC)*. Malacca, Malaysia: IEEE, Dec. 2022, pp. 274–279. ISBN: 978-1-66547-098-8. DOI: [10.1109/ICSPC55597.2022.10001803](https://doi.org/10.1109/ICSPC55597.2022.10001803). URL: <https://ieeexplore.ieee.org/document/10001803/> (visited on 08/18/2023) (cit. on p. 60).
- [72] W. Yuan, Z. Li, C.-Y. Su. “Multisensor-Based Navigation and Control of a Mobile Service Robot”. en. In: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 51.4 (Apr. 2021), pp. 2624–2634. ISSN: 2168-2216, 2168-2232. DOI: [10.1109/TSMC.2019.2916932](https://doi.org/10.1109/TSMC.2019.2916932). URL: <https://ieeexplore.ieee.org/document/8727415/> (visited on 08/18/2023) (cit. on p. 36).
- [73] K. Zheng. *ROS Navigation Tuning Guide SLAM*. URL: [https://kaiyuzheng.me/documents/papers/ros\\_navguide.pdf](https://kaiyuzheng.me/documents/papers/ros_navguide.pdf) (cit. on p. 61).



### **Declaration**

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature