

UNIVERSITY OF TARTU
Faculty of Science and Technology
Institute of Bioengineering

Omar Huseynli

ROS integration for the Semubot robot

Bachelor's Thesis (12 ECTS)

Curriculum Science & Technology

Supervisor:

Renno Raudmäe, MSc

Tartu 2025

ROS integration for the Semubot robot

Abstract

Humanoid robots are increasingly important in various fields like healthcare, industrial automation, and human-assistive technologies. In healthcare, humanoid robots are useful in assisting individuals with special needs, particularly children with Autism Spectrum Disorder (ASD). [1] These social robots enhance social interaction, communication skills, and learning engagement. Humanoid robots help children to develop eye contact, emotional recognition, and communication skills through structured human-robot interaction. In this thesis work, I aim to develop a Robot Operating System framework for Semubot, the first humanoid robot in Estonia.

Keywords

Semubot, humanoid robots, Robot Operating System, ASD

CERCS: T125; Automation, robotics, control engineering

Institute name: Institute of Bioengineering

Research group: Intelligent Materials and Systems Lab, Institute of Technology

ROSi võimekuse lisamine Semuboti robotile

Lühikokkuvõte

Humanoidrobotid on üha olulisemad erinevates valdkondades, nagu tervishoid, tööstusautomaatika ja inimest abistavad tehnoloogiad. Tervishoius on humanoidrobotid kasulikud erivajadustega inimeste, eriti autismispektri häirega (ASD) laste abistamiseks. [\[1\]](#) Need sotsiaalsed robotid parandavad sotsiaalset suhtlust, suhtlemisoskusi ja õppimisse kaasatust. Humanoidrobotid aitavad lastel arendada silmsidet, emotsionaalset äratundmist ja suhtlemisoskusi läbi struktureeritud inimese ja roboti suhtluse. Käesoleva lõputöö eesmärgiks on välja töötada robotite operatsioonisüsteemi raamistik Eesti esimesele humanoidrobotile Semubot.

Võtmesõnad:

Semubot, humanoidrobotid, robotite operatsioonisüsteem, ASD

CERCS: T125; Automatiseerimine, robotika, juhtimistehnika

TABLE OF CONTENTS

TERMS, ABBREVIATIONS, AND NOTATIONS.....	5
INTRODUCTION.....	6
LITERATURE REVIEW.....	7
1.1 Introduction to Literature Review.....	7
1.2 Humanoid robots in healthcare.....	7
1.3 The role of ROS in humanoid robots.....	8
1.3.1. Android-based Middleware.....	9
1.3.2 Hybrid Middleware: ROS with NAOqi in Pepper robot.....	9
1.3.3 YARP middleware.....	10
1.3.4 CogMap and DiODE: Middleware in ASIMO humanoid robot.....	11
1.3.5 Robot Operating System 2.....	12
1.3.6 Comparison of Middleware for Humanoid Robots.....	13
1.4 Digital Twin Technology in Humanoid Robots.....	14
2 THE AIMS OF THE THESIS.....	17
3 RESEARCH AND DEVELOPMENT.....	18
3.1 ROS2 Environment Setup.....	18
3.2 Robot Description: URDF Modelling.....	18
3.3 Command Velocity Implementation.....	22
3.4 Motor Control Integration via ROS2 on Semubot.....	23
3.5 Integration in CoppeliaSim (V-REP).....	24
3.6 RESULTS.....	26
3.7 Challenges.....	28
3.8 DISCUSSION.....	28
SUMMARY AND FUTURE WORK.....	29
Future Work.....	29
REFERENCES.....	32

TERMS, ABBREVIATIONS, AND NOTATIONS

AI - Artificial Intelligence

API - Application Programming Interface

DDS - Data Distribution Service

GPIO - General Purpose Input Output

GUI - Graphical User Interface

HD - High Definition

HRI - Human Robot Interaction

IDL - Interface Definition Language

IP - Internet Protocol

OS - Operating System

PWM - Pulse Width Modulation

ROS - Robot Operating System

SD - Secure Digital

SSH - Secure Shell Protocol

SLAM - Simultaneous Localization and Mapping

SWIG - Simplified Wrapper and Interface Generator

UI - User Interface

URDF - Unified Robot Description Format

YARP - Yet Another Robot Platform

3D - Three-Dimensional

INTRODUCTION

Humanoid robotics is one of the most dynamic fields in robotics development. Because of their engineering human-like design, humanoid robots can easily interact with people in human environments. These robots could be employed in elderly companionship, cognitive training, rehabilitation, patient motivation through athletics, and ergotherapy because they can transmit nonverbal and verbal communication through voice, gestures, and the most advanced form of technology today, AI (artificial intelligence) [2]. Unlike traditional machines, these robots interact on a completely different level, not only catering to the physical but also emotional and psychological requirements of their patients. This exceptional characteristic makes humanoid robots useful for long-term care patients who do not get much attention from caregivers. Attendant care is in high demand, and in conjunction with the world's increasing interest in socially assistive technologies, healthcare systems are considering the application of humanoid robots to relieve staff workload while improving patient care and outcomes.

With the growing interest in humanoid robots, there is a growing demand for middleware that is not only capable but also modular and flexible. However, many of today's available humanoid robots rely on closed-source middleware. It limits their customization, scalability, and opportunity to integrate with new technologies.

To handle these limitations, the ROS (Robot Operating System), an open-source framework that provides tools, libraries for building robot applications, has been adopted by the robotics community. The latest version, ROS2, has critical improvements such as secure communication, support for distributed systems. These features make ROS2 more suitable for modern robotics projects.

Despite the benefits offered by ROS2, its application in humanoid robotics is still relatively limited. Most ROS2 applications focus on mobile robots or industrial arms, while humanoid robots still use proprietary closed-source frameworks.

In this thesis, I will explore the application of ROS2 in a humanoid robot named Semubot, the first humanoid robot developed in Estonia. The main objective is to implement the ROS2-based middleware framework for Semubot. By focusing on open-source tools, this project aims to create a foundation for further research and development on Semubot, contributing to developing an affordable and adaptable humanoid robot platform in healthcare.

LITERATURE REVIEW

1.1 Introduction to Literature Review

This literature review explores recent studies on humanoid robots, especially in healthcare. It highlights the role of ROS in enabling modular robotics development and explores the potential of Digital Twin Technology for simulation and testing. The main goal is to uncover relevant studies on middleware-related development and gaps that inform the design of the Semubot humanoid robot.

1.2 Humanoid robots in healthcare

Pepper ([Figure 1.1](#)), a humanoid robot made by SoftBank Robotics, has been extensively used for dementia care, rehabilitation therapies at home, as well as educational interaction with children. Providing emotional support, it has easily worked with patients [\[3\]](#). For elderly patients with Mild Cognitive Impairment, it has been used in Cognitive Stimulation Therapy and serves as a social companion, thus helping in keeping the patients engaged [\[3\]](#). Alongside ChatGPT, Pepper has helped patients with Autism Spectrum Disorder by providing real-time conversational exchange, which was more flexible and improved emotional and social communication [\[4\]](#). Pepper has also been used in long-term care facilities during the COVID-19 pandemic to help caregivers with automated health checks ([Figure 1.2](#)) [\[5\]](#). It has further served in children's diabetes education, helping them reduce the stress around chronic cases by gamified interaction, which improves retention of knowledge [\[6\]](#).

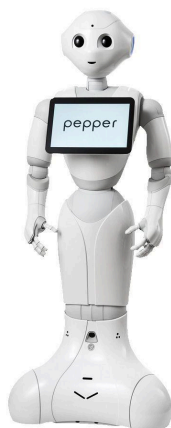


Figure 1. - 1. Pepper Robot [\[7\]](#) 2. Pepper has served as a health screening assistant in long-term care facilities, particularly during the staffing shortages brought on by the COVID-19 pandemic [\[5\]](#).

NAO ([Figure 2](#)) is another robot that assists in healthcare. NAO has adaptive behavior depending on user interaction, which enables personalized education and therapy employing reinforcement learning techniques [\[8\]](#). NAO has also helped with the care of elderly people by facilitating and aiding them to participate in more memory and group activities, which has been shown to reduce the isolation of the elderly and encourage them to engage more socially [\[8\]](#).



Figure 2. NAO robot [\[9\]](#)

1.3 The role of ROS in humanoid robots

Middleware is defined as “a class of software technologies designed to help manage the complexity and heterogeneity inherent in distributed systems” and “a layer of software above the operating system but below the application program that provides a common programming abstraction across a distributed system” [\[10\]](#). In robotics, middleware connects hardware such as sensors, actuators, and controllers using the robot’s application logic, managing communication between them. This enables building complex robotics applications more efficiently without the need to interact with low-level hardware logic [\[11\]](#).

ROS is an open-source software development tool that serves as a middleware framework for research and development in robotics [\[12\]](#). Unlike a traditional operating system, ROS is a software architecture with a robust set of tools, libraries, and necessary components for communication that help in the development of robotic applications. Within its modular

design, components “nodes” interface with each other via topics, services, and actions. Through hardware abstraction, ROS simplifies integration of sensors, motion planning, mapping, and even simulation, which allows easy construction of advanced robotic systems with little low-level code [12].

To enable movement, perception, and interaction in modern humanoid robots, a multitude of control systems are utilized. The upcoming sections analyze the control systems deployed by Pepper, NAO, ASIMO, and others, emphasizing in-house developments as well as the increasing use of ROS-based systems.

1.3.1. Android-based Middleware

Advancements in Pepper control architecture have recently increased the robot’s versatility and ease of use in the healthcare industry. Pepper’s capabilities have greatly improved with new Android-based middleware that integrates via WebSocket servers and Python scripting, replacing the proprietary NAOqi framework [2]. This middleware allows non-experts to configure Pepper for various clinical settings with functions such as text-to-speech, navigation, and barcode scanning.

Pepper’s hardware includes an HD (High Definition) camera, 3D vision sensors, tactile and position sensors, wheels with multidirectional movement, and programmability in C++, Python, and Java [2].

1.3.2 Hybrid Middleware: ROS with NAOqi in Pepper robot

NAOqi is middleware software developed by SoftBank Robotics to control such humanoid robots as NAO and Pepper [13]. It serves as the main API (Application Programming Interface) [14], handling communication between software and hardware components. One of the best advantages of NAOqi is that it is not difficult to integrate into a robotics project, so non-technical users can manage it. The framework is very useful in research and educational environments because it is based on Python, which lets users quickly build and test interaction scenarios [13].

Pepper robot's autonomous navigation and personalized interaction functionalities were improved through integration with ROS, which augments the proprietary NAOqi operating system [15]. In this setup (Figure 3), ROS takes care of SLAM (Simultaneous Localization and Mapping) [16], while NAOqi manages control of the functions, movements, and access to the sensors’ data. The integration is done through a dedicated ROS Interface node, which translates between ROS topics and NAOqi API calls, so Pepper can subscribe to odometry

and depth images and simulate commanded laser scans, and respond to ROS velocity commands [15].

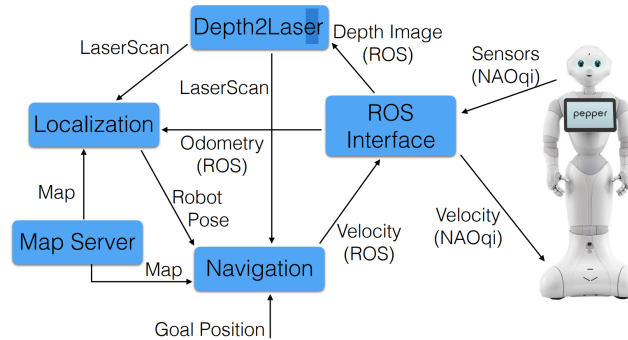


Figure 3. Software architecture for localization and navigation with ROS in the Pepper robot, with ROS nodes shown in blue, and arrows representing communication over topics [15].

With this setup, Pepper can interact and navigate through dynamic environments using a depth sensor that functions as a laser, speech, face recognition, and tablets for user interfacing. These interaction capabilities are enhanced through IBM Watson cloud services for speech recognition and custom web services for face recognition. The entirety of the ROS stack, including real-time navigation modules, was installed on the robot's virtual machine running an emulated version of NAOqi OS (Operating System) so that the machine appears to ROS, creating an isolated development environment.

1.3.3 YARP middleware

YARP (Yet Another Robot Platform) is an open-source, lightweight middleware for real-time robotic systems that operates on humanoid robots like iCub [17]. First, YARP was tested on the robot Kismet. YARP is capable of enabling peer-to-peer interaction via TCP, UDP, and shared memory, among others. One of YARP's strengths is its extremely non-intrusive architecture, which operates more like a library than a framework. This permits system developers to effortlessly plug YARP into a wide array of systems.

Despite providing such benefits, YARP has significant limitations. The absence of IDL (Interface Definition Language) [18] forces formatting of messages to be manual, making complex remote procedure calls inefficient in comparison to other systems such as ROS. Other factors that inhibit YARP are the C++ core with other language bindings through SWIG (Simplified Wrapper and Interface Generator) [19], which makes seamless integration

challenging in practice [17]. Its reliance on a centralized naming server also imposes the risk of a single failure point within distributed applications.

Even with these challenges, YARP is already used in such robots as Obrero, Eurobot, Babybot, Domo [20].

1.3.4 CogMap and DiODE: Middleware in ASIMO humanoid robot

The ASIMO humanoid robot from Honda has a distributed system design; it uses communication middleware to mediate the exchange of information between software modules as well as the hardware components [21]. This middleware (Figure 4) has three modules: CogMap (Cognitive Map), DiODE (Distributed Operation via Discrete Events), and MC (Multimodal Communication).

CogMap is an architecture with multimedia modules and enables active transformation of information streams using asynchronous message passing dynamic inter-module links.

DiODE brings two modules together with a cable and socket, providing a strong link. The module's communication device is known as a port.

MC works as a video server that delivers video transport streams over IP (Internet Protocol) in response to requests for remote processing, enabling distributed system computation.

Test results showed that with sufficient network bandwidth, middleware delays are insignificant compared to the module's computation time [21].

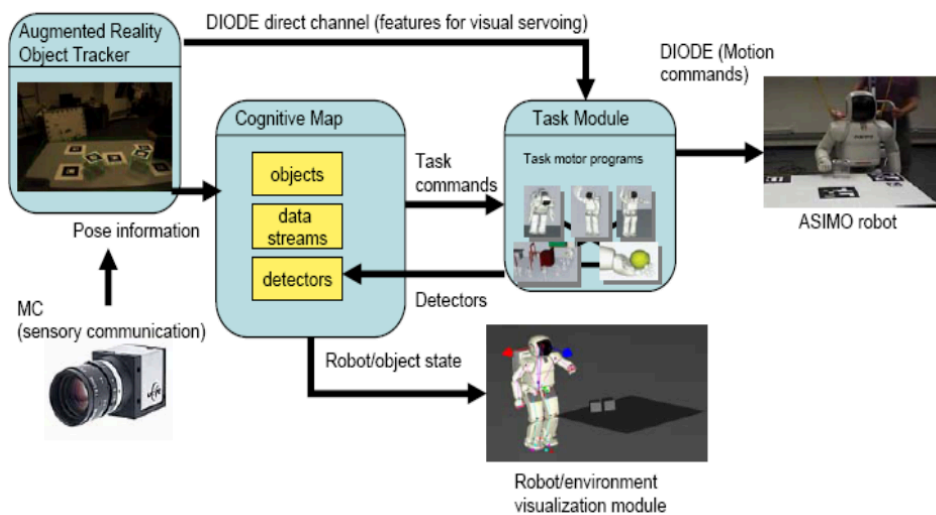


Figure 4. Modular middleware architecture used in ASIMO, showing the interaction between sensory communication (MC), cognitive processing (Cognitive Map), task execution (Task Module), and robot control through DiODE [\[21\]](#).

1.3.5 Robot Operating System 2

The ROS2, an updated version of ROS, is now more reliable and flexible for developing robotic systems [\[22\]](#). It introduces several improvements, such as DDS (Data Distribution Service) [\[23\]](#) for more reliable communication, and adds built-in security features like authentication, encryption, and access control [\[24\]](#), which were absent in ROS1.

Additionally, supporting multi-robot communication [\[22\]](#) makes ROS2 appropriate for autonomous vehicles and robots in industry.

Use of Robot Operating System 2 in the NAO humanoid robot

Adoption of ROS2 in the NAO humanoid robot marks an important milestone in making the robot more autonomous, adaptable, and simpler for HRI (Human Robot Interaction) applications development [\[25\]](#). OAN, a framework operated on ROS2, supersedes the external bridges to NAOqi middleware's proprietary framework by executing ROS2 on a custom Ubuntu system within NAO. This framework decreases latency, increasing access to different hardware parts. Such a configuration augments NAO's capabilities by enabling real-time control of motion, speech synthesis and recognition, vision processing with YOLO [\[26\]](#), and conversing through generative models.

1.3.6 Comparison of Middleware for Humanoid Robots

From the middleware platforms discussed above, it is clear that every system has noted benefits in regard to specific use cases in humanoid robotics. Nonetheless, when analyzing the criteria of scalability, real-time responsive interaction, safety measures, system modularity, and support from the user base, it's evident that ROS2 stands out as the most versatile and reliable option. Table 1 below compares and summarizes the aforementioned middleware options by detailing their respective advantages and disadvantages.

Table 1. Comparison of Robot Middleware Platforms

Middleware	Used in	Strengths	Weaknesses	ROS2 advantage
Android-based	Pepper	Simple UI, easy python scripting	Limited to pepper, lacks modularity	Not scalable
NAOqi	NAO, pepper	Easy to use, good for education and HRI	Closed-source, outdated, no real-time support	Outdated and closed-source
Hybrid ROS+NAOqi	Pepper	Enables ROS features, saving NAOqi APIs	Complex bridging, increased latency	Transitional setup
YARP	iCub, Kismet, Babybot	Peer-to-peer, decentralized, lightweight	Complex message handling, weak multi-language integration	Integration overhead
CogMap/DiODE	ASIMO (Honda)	Custom, distributed, real-time	Proprietary, not reusable, lacks community support	Non-transferable
ROS1	Turtlebot, PAL robots	Open-source, modular, rich ecosystem	No real-time support, lacks security	Predecessor of ROS2
ROS2	Modern NAO robot, PAL robots	Real-time DDS, secure, scalable, ROS1-compatible	Requires complex setup knowledge	Best overall platform

Based on the comparison table ([Table 1](#)), it is apparent that ROS2 is the most appropriate middleware for the development of Semubot. This is due to the fact that it is open-sourced, which means it is available and can be modified freely. Moreover, having real-time capabilities, DDS-based communication, multi-robot systems, and advanced security features makes it suitable for modular robot systems.

In addition, ROS2 is supported by a wide and well-organized community of developers, which provides a lot of active documentation, reproducible libraries and packages, and an evolving ecosystem that is helpful to new programmers. These reasons will lower the effort and time invested considerably in development planning, perception, simulation design, adding functionality to add the motion, implementing the Digital Twin system for Semubot, and many more.

1.4 Digital Twin Technology in Humanoid Robots

Extensive evaluation is often conducted before deploying humanoid robots in the real world to make sure that they are safe and reliable to use. This step would require physical testing, which poses a number of cost-losing risks, and thus robots are first tested on simulators, which allows them to be optimized in a safe virtual environment [\[27\]](#).

Physical robots can now be trialed through the use of Digital Twin technology simulators. This is one of the most advanced approaches to simulation, which creates replicas in real time, guaranteeing accuracy, down to the actuators, sensors, and joints.

To enable effective implementation of Digital Twin Technology, we use several simulators such as WeBots [\[28\]](#), CoppeliaSim (formerly V-REP) [\[29\]](#), and Gazebo [\[30\]](#). These simulators are integrable with ROS, so that we can transfer commands to both real and virtual robots, ensuring that the same software is running in a real and simulated environment at the same time.

In evaluating the performance of CoppeliaSim, Gazebo, and WeBots simulators for the NAO robot, an experiment was done [\[31\]](#). The robot had to navigate to a goal position and avoid an obstacle, such as a chair, placed between its starting point and the goal ([Figure 5](#)). The robot was required to maneuver around the chair to complete the task under conditions involving a high number of joint movements necessary for obstacle avoidance, to compare the

performance of simulators. The study focused on measuring CPU usage, memory footprint, and disk access during task execution.

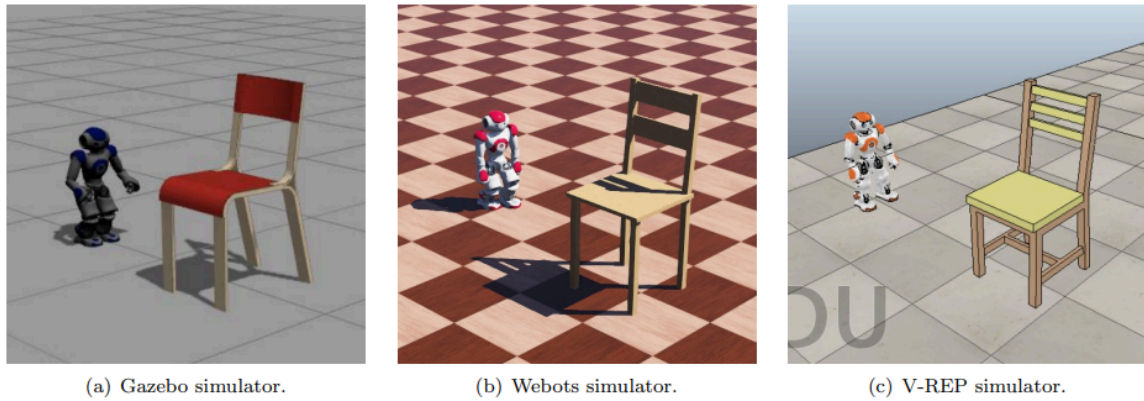


Figure 5. Simulation environments in Gazebo, Webots, and V-REP with a NAO robot avoiding an obstacle (a chair) [\[31\]](#).

Results showed that WeBots is the most resource-efficient simulator, while Gazebo consumed most of the CPU because of its client-server architecture. CoppeliaSim showed stable CPU usage, while the CPU usage of Gazebo and WeBots was fluctuating. Regarding disk access, WeBots is the most lightweight, being useful for non-complex tasks, while Gazebo and CoppeliaSim showed similar higher disk space usage. As a result, CoppeliaSim offers stable memory usage, WeBots is excellent for simple tasks, and Gazebo is the best for complex projects [\[31\]](#).

Moreover, to select the best simulator, such factors as ROS integration, sensor modelling, and ease of use for students must be considered [32]. The following table compares the simulators we talked about above, highlighting pros and cons:

Table 2. Comparison of CoppeliaSim, Gazebo and WeBots based on research [32].

Platform/ characteristics	CoppeliaSim	Gazebo	Webots
Integration with ROS	Full integration	Full integration	Integration at average level
Ease of use	High entry threshold, but great opportunities	Average entry threshold	High, especially for educational purposes
Graphical interface	Intuitive and rich in features. Flexible interface with many customizable windows.	Simple, focused on efficiency. Minimalistic interface focused on 3D visualization.	Simple, focused on efficiency. Minimalistic interface focused on 3D visualization.
Licensing	Commercial and free versions	Open source (Apache 2.0)	Commercial version, free for academic use
Modeling of sensors	Variety of sensors with high level of customization.	Wide range of sensors with possibility of expansion.	Large number of ready-made sensors.
Modeling of physics	Bullet, ODE, Newton, Vortex	Bullet, DART, ODE, Simbody	Bullet, ODE.
Supported operating systems	Windows, macOS, Linux	Windows, Linux, macOS (through WSL)	Windows, macOS, Linux
Support for programming languages	C/C++, Java, Python, Lua	C++, Python, ROS	C, C++, Java, Python, Matlab

Based on the comparison table (Table 2), we can consider CoppeliaSim as the most suitable simulator for the Semubot project. Its full integration with ROS2, GUI (Graphical User Interface), high ease of use, and wide variety of sensor and physics modelling make it ideal for developing and simulating the Semubot project.

2 THE AIMS OF THE THESIS

- Aim 1: To develop a ROS2-based framework for the Semubot humanoid robot.
- Aim 2: To implement a Digital Twin of Semubot using CoppeliaSim (V-REP).
- Aim 3: To get the Robot moving by using ROS2 built-in libraries

3 RESEARCH AND DEVELOPMENT

3.1 ROS2 Environment Setup

To run this Semubot project, the Robot Operating System 2 (ROS2) distribution Jazzy was installed on a machine running Ubuntu 24.04. Ubuntu is a linux-based operating system that was chosen because of its compatibility with ROS2 Jazzy and its stable kernel. ROS2 is an open-source framework for developing modular robot software. The full environment setup, including installation and dependencies, is available in the project's GitHub repository [\[Appendix 1\]](#).

3.2 Robot Description: URDF Modelling

The Unified Robot Description Format (URDF) is an XML-based specification used in ROS2 to represent a robot's physical structure and kinematic structure in a simulator [\[33\]](#). It provides standardized ways to describe a robot's links (rigid parts) and robot's joints (movable connections), including visual, collision, and inertial properties. In the context of our ROS2 project, URDF is important because it allows such visualization tools as RViz2 and simulators, such as Gazebo or CoppeliaSim (V-REP), to show a robot's 3D structured model.

In this project, the URDF file 'semubot.urdf' is the foundation of the Semubot robot. It defines a robot's geometry using STL meshes (stored in the urdf/ folder), assigns coordinate frames to each part, and establishes how parts are connected through joints. The URDF is loaded during launch using the robot_state_publisher node, which broadcasts the TF (transform) tree that shows how each part of the robot is spatially related to others. By this, real-time visualization of the robot's configuration is achieved inside RViz2. RViz2 [\[34\]](#) is a powerful 3D visualization tool in ROS2 to observe robot models, sensor data, and coordinate frames (TF), and topic messages in real-time. By this, we can check how a robot is

configured, and whether messages are correctly affecting the robot's behavior.

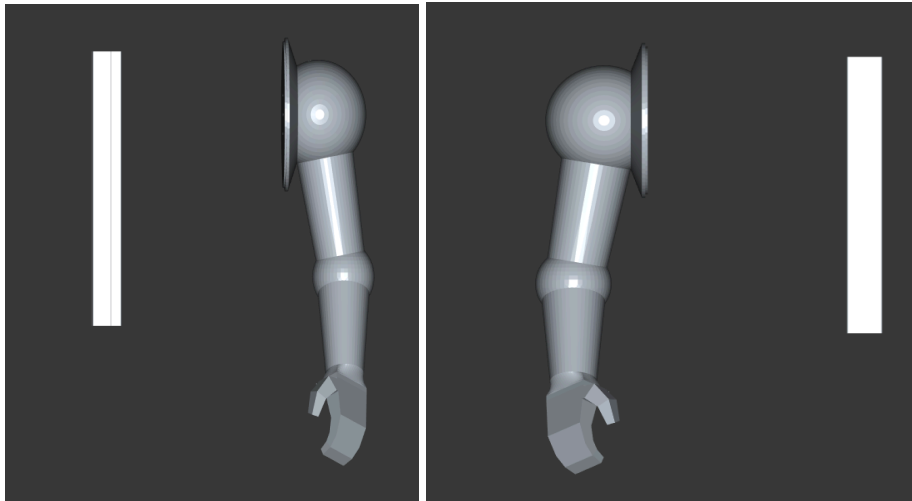


Figure 6. - 1. Left arm of Semubot, 2. Right arm of Semubot.

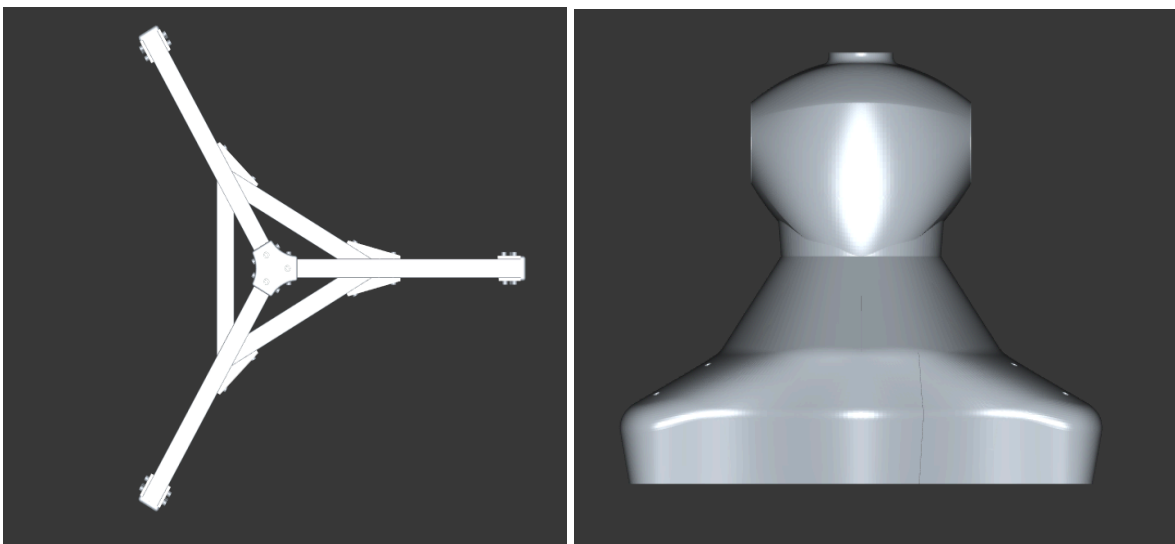


Figure 7. - 1. Movement frame of Semubot, 2. Main body of Semubot.

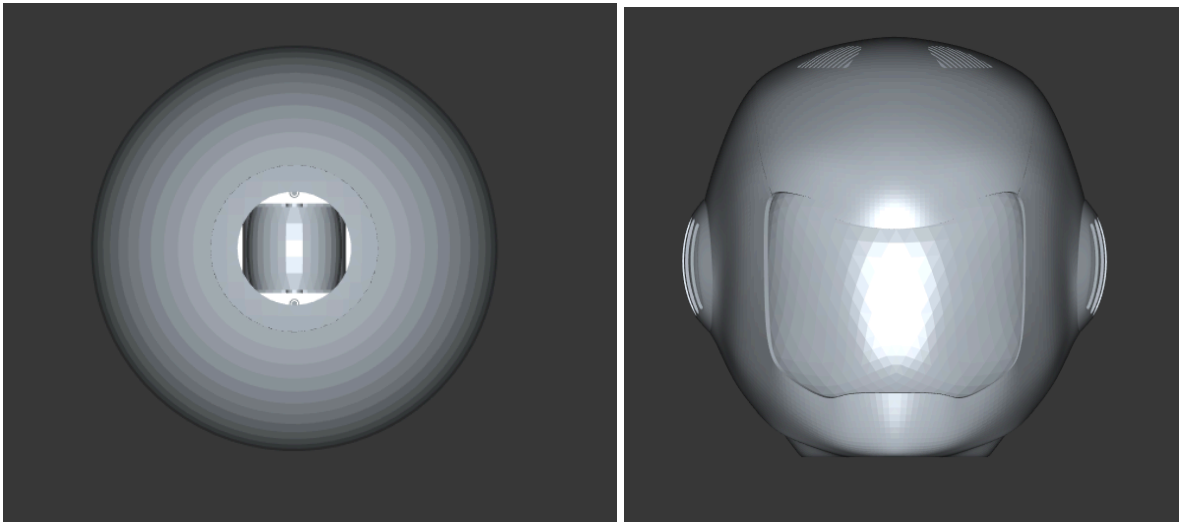


Figure 8. - 1. Omniball of Semubot, 2. Head of Semubot.

In `semubot.urdf` each component of the robot - left and right arms ([Figure 6.1, 6.2](#)), movement frame ([Figure 7.1](#)), body ([Figure 7.2](#)), omniballs ([Figure 8.1](#)), head ([Figure 8.2](#)) - is modeled as a link, and connections between them are defined as joints. In any URDF file, `base_link` must be defined. The `base_link` is the root reference frame of a robot and plays the role of a central coordinate system in which all other components are connected. The movement frame was chosen as `base_link` because it is the lowest and most stable part of the robot that interfaces with the ground. Head and arms are defined as links according to Semubot's body part, while omniballs and the body of Semubot are defined as links according to the movement frame. The head joint was defined as a rotational (revolute) joint to allow head movement in a limited range. Both the left and right arms are connected with continuous joints to enable free 360-degree motion. Omniballs are also attached to the movement frame using continuous joints to support free movement.

3.21 Visualization of Semubot URDF in RViz2

To visualize our semubot.urdf, we must write a launch file in python script. This launch file starts with three essential components: the joint_state_publisher node, which provides joint angle information, the robot_state_publisher node, which broadcasts coordinate transforms (TF) based on the URDF, and RViz2, which enables visualization of the robot. From [Figure 9](#), you can see the first successful test run of the Robot in the RViz2 environment.

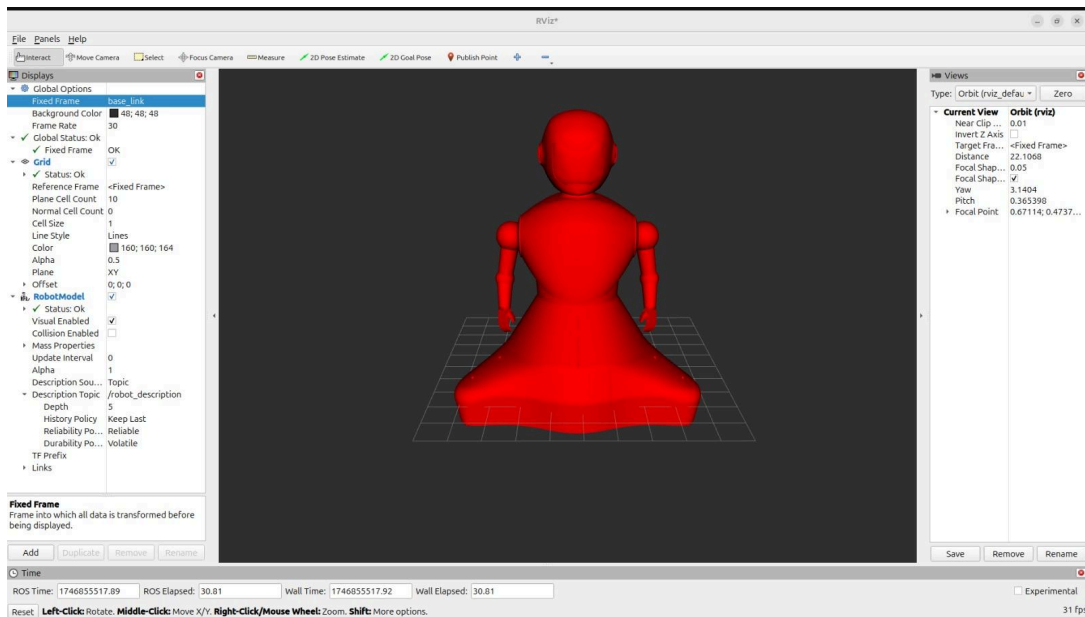


Figure 9. Visualization of Semubot in RViz2.

Rqt_graph is a useful ROS2 tool that shows all nodes, their connections through topics, helping us to understand what happens. After visualizing Semubot URDF in RViz2, by running rqt_graph we can be sure that all nodes and topics work correctly ([Figure 10](#)).

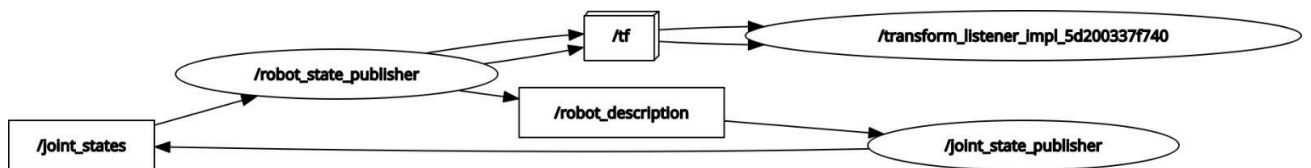


Figure 10. Rqt_graph shows that topics such as /robot_description, /tf, and /joint_states are connected correctly to nodes.

3.3 Command Velocity Implementation

The `/cmd_vel` topic in ROS2 is a built-in interface for controlling robot movement by using velocity commands. It carries `geometry_msgs/msg/Twist` type messages, which include both linear and angular velocities in x, y, z, in other words, three-dimensional space. In mobile robotics, including Semubot, only linear.x and angular.z velocities are typically enough. Linear.x velocity is used for backward (negative value) or forward movement(positive value), while angular.z is used for the robot's rotation around the vertical axis. These velocities must be published to `/cmd_vel` topic, so that other nodes can subscribe and use these velocities. Moreover, `/cmd_vel` topic allows developers to control how the robot moves without directly interacting with motor controllers because motor controllers are used by completely different subscriber-nodes. So, the main advantage is that nodes responsible for publishing velocities to `/cmd_vel` topic are not hardcoded in case motor controllers are changed. In the Semubot project, `/cmd_vel` topic is the main topic for movement commands linking velocities to both physical and simulated robots.

`/cmd_vel` topic in Semubot uses several ROS2 nodes working together:

- `cmd_vel_publisher`
- `cmd_vel_to_joint_states`
- `odom_broadcaster`

The `cmd_vel_publisher` node periodically sends velocity commands in the form of `geometry_msgs/Twist` messages. In this setup, `/cmd_vel` topic receives linear velocity on x.axis (-0.5 m/s) and zero angular velocity. These velocities simulate backward movement. This message is published on the `/cmd_vel` topic at 1 Hz frequency.

The `cmd_vel_to_joint_states` node subscribes to `/cmd_vel` topic. This node is responsible for omniball joint movement by integrating velocity over time to predict joint position. This node creates a `sensor_msgs/JointState` message, after it publishes it to `/cmd_vel_to_joint_states` topic. After the `joint_state_publisher` node subscribes to `/cmd_vel_to_joint_states` topic, we can see changes in the joints' movement.

There is the formula I used to transform velocity to position, so that the `joint_state_publisher` node understands it:

$$\text{wheel_position} = (\text{linear_velocity} \pm \text{angular_velocity}) * \text{delta_t}$$

In the formula above, `delta_t` is the time between updates. Depending on each omniballs's behavior, we either subtract or add angular velocity to copy the real omniball movement.

The odom_broadcaster node calculates the robot's odometry using the same /cmd_vel topic. It calculates x and y position, and orientation based on the following formulas:

$$\Delta x = (v_x * \cos(\theta) - v_y * \sin(\theta)) * \Delta t$$

This formula calculates how much the robot moves toward X direction, considering its current orientation (theta).

- v_x is linear velocity along the X axis.
- v_y is usually 0 for most robots, but it was kept in this formula because of Semubot's omnidirectional movement.
- $\cos(\theta)$ and $\sin(\theta)$ rotate movement from robot frame to global frame.

$$\Delta y = (v_x * \sin(\theta) + v_y * \cos(\theta)) * \Delta t$$

This formula has the same idea as the formula above, but this calculates movement in the Y direction.

$$\Delta \theta = \omega * \Delta t$$

- This formula updates the robot's orientation.
- ω (omega) is the angular velocity around the Z axis (using angular.z in /cmd_vel).
- It is multiplied by time to get how much the robot turned from the last update.

The node creates a continuous transform between the odom frame (fixed world reference) and the base_link (Semubot's body). In more detail, this node subscribes to /cmd_vel topic and listens for linear and angular velocity commands. On each callback, it calculates how much time has passed since the last update using the system clock. Using the formulas explained above, it converts velocity to the new position and orientation. After, it creates a TransformStamped message, sets odom as parent frame and base_link as child frame, and sends it using TransformBroadcaster. As a result, we can see the robot's movement in the RViz2 visualization tool.

3.4 Motor Control Integration via ROS2 on Semubot

The current control mechanism of Semubot, previously developed by Georgs Narbutis [35], is based on Raspberry Pi 4 Model B [36], which acts as the “brain” of the Semubot wheelbase. The Raspberry Pi is connected to three Brushless DC motor drivers (BLD-510S) [37]. Each of these drivers is responsible for controlling the 57BLR70-24-02-HG50 BLDC [38] motor installed on Semubot's movement frame. The Raspberry Pi 4B, with its quad-core ARM Cortex-A72 processor and integrated GPIO support, provides a compact platform for low-level motor control. GPIO (General Purpose input/output) is a set of programmable pins

on devices, including Raspberry Pi 4B, that can be used to read inputs (for example, from sensors) and send outputs, in our case, to motors. It facilitates the process of developing software for direct interaction with hardware components.

In the initial version of the system, motor control was handled by a Python script running on Raspberry Pi OS. This script used the RPI.GPIO library to manage PWM signal and direction control for each motor based on the joystick input. The script included a kinematic matrix to translate directional input into corresponding wheel velocities. While functional, this approach lacked integration with the broader ROS2 architecture.

To make the transition to the ROS2 environment on Semubot, a new SD card with Ubuntu 24.04 was flashed on the Raspberry Pi, because the previous Raspberry Pi OS was not compatible with ROS2 Jazzy. After ROS2 Jazzy and the environment were installed and configured through SSH (Secure Shell Protocol) to the RPi.

Based on the original Python motor control script, ROS2 named node `cmd_vel_motor_driver` was developed. This node used previous GPIO-based motor control logic, but was modified to subscribe to `/cmd_vel` topic using `rclpy` library. By this configuration, Raspberry Pi received velocity commands (linear.x and angular.z) over the network in real time.

Main ROS2 setup, including `cmd_vel_publisher`, robot simulation and RViz2 visualization, was running on an external laptop. On Pi's ROS2 workspace named `cmd_vel_motor_driver` node was configured. To connect these two systems and make real-time communication, ROS2 built-in Discovery tool over the local network was implemented. Because both laptop and Pi were connected to the same network, ROS2 DDS (Data Distribution Service) discovery protocol automatically detected laptop and RPi, and made communication between them. As mentioned before, DDS is one of the main advantages of ROS2 over previous ROS1.

After configuring ROS2 discovery protocol, launch file on external laptop and subscriber node on Pi were run. As a result, Raspberry Pi successfully subscribed to `/cmd_vel` topic on an external laptop, and translated these velocities into motor signals using GPIO. This completed the goal of synchronized simulation in RViz2 and real-world movement of Semubot humanoid robot.

3.5 Integration in CoppeliaSim (V-REP)

As a part of this project, effort was made to simulate Semubot in the CoppeliaSim environment. CoppeliaSim is a widely used robot simulator, supporting integration using the ROS2 plugin. The goal was to import Semubot robot, connect it to the `/cmd_vel` topic for

movement commands and publishing robot's real-time position back to ROS2 as a transform from odom (fixed world reference) to base_link (robot's center of motion). This would allow full bidirectional integration, realizing goal as a Digital Twin.

The transform odom to base_link in ROS2 is widely used in mobile robot localization to track robot's position over time. Publishing this transformation from CoppeliaSim to ROS2 enables accurate tracking of the robot's movement in the virtual world and synchronizes it with ROS2 to the real world.

Semubot's URDF file was successfully imported in CoppeliaSim using the built-in URDF Importer tool, which generated Semubot's physical model in simulation ([Figure 11](#)). The STL meshes, robot's components, were correctly scaled and placed. Furthermore, the CoppeliaSim ROS2 interface plugin was enabled, and ROS2 discovery protocol confirmed communication with other ROS2 nodes (/sim_ros2_interface).

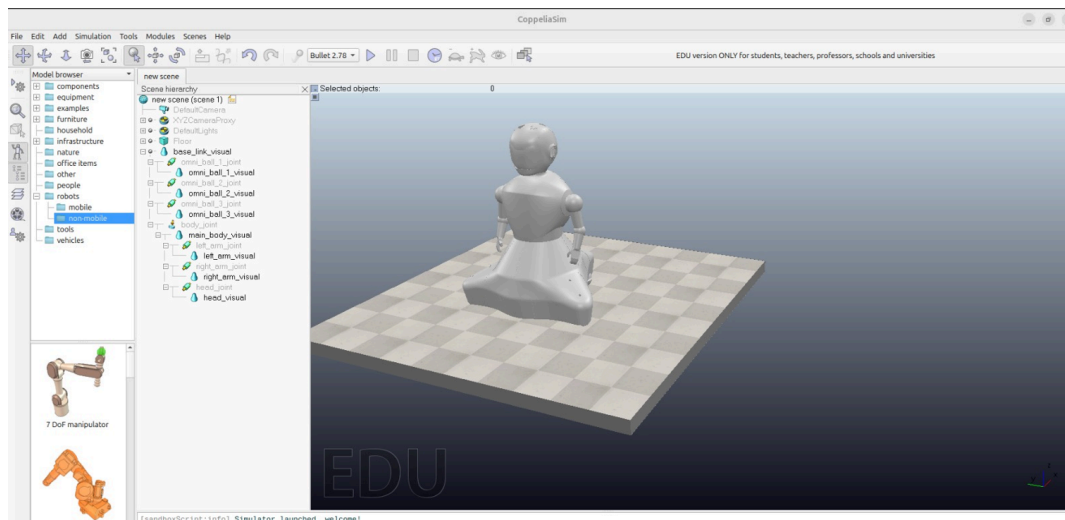


Figure 11. Successfully imported Semubot's URDF in CoppeliaSim after enabling the ROS2 interface plugin.

In CoppeliaSim to run simulation, Lua script must be written and attached to the correct object of simulation. A custom non-threaded Lua script was attached to the robot's baselink. It was supposed that this script will subscribe to /cmd_vel topic and apply received velocities to the robot. However, despite no error in script after running, the robot did not show any movement during simulation, even after setting the robot model as dynamic and enabling its gravity.

3.6 RESULTS

The Semubot humanoid robot was successfully modeled and controlled using ROS2 Jazzy middleware. The URDF file defined the physical structure of Semubot, which included its arms, head, movement frame, and omniballs, using STL files. By launching through a custom launch file, the robot was correctly visualized in RViz2. It showed all links and joints connected by TF.

Command velocity control was also successfully implemented using ROS2 `/cmd_vel` topic. The node `/cmd_vel_publisher` generated velocity commands, while `/cmd_vel_to_joint_states` subscribed to this topic, published joint positions, simulating movement of omniballs. Moreover, the `odom_broadcaster` node calculated the robot's position based on published velocities. After using formulas, it enabled root movement. As a result, RViz2 successfully visualized omniballs' and robot's movement ([Figure 12](#)).

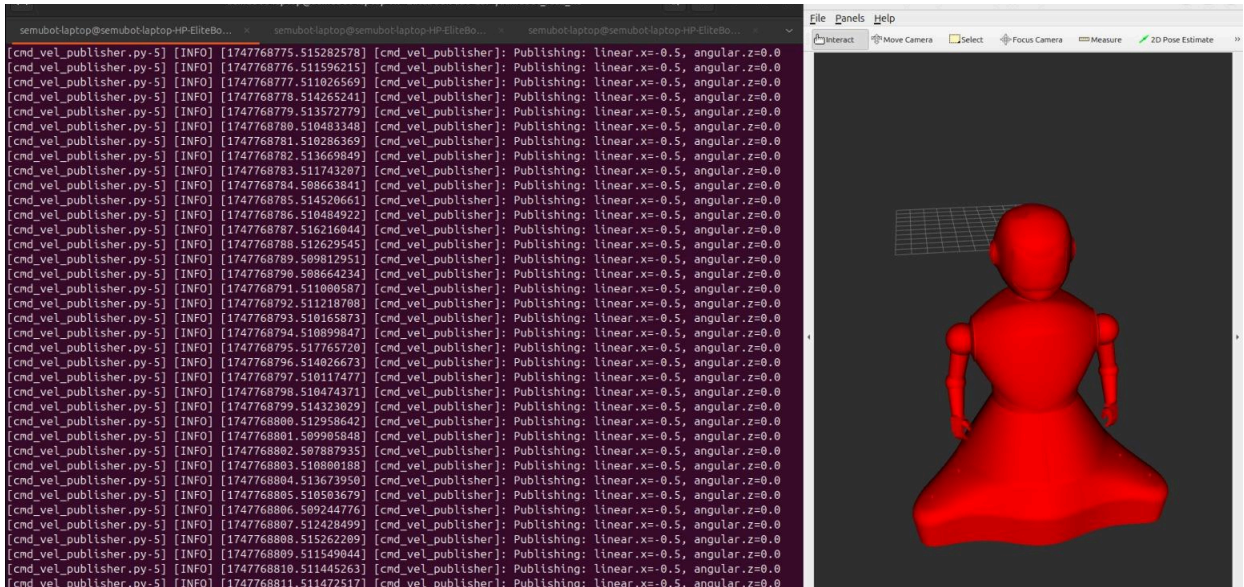


Figure 12. Successful visualization of moving Semubot in RViz2.

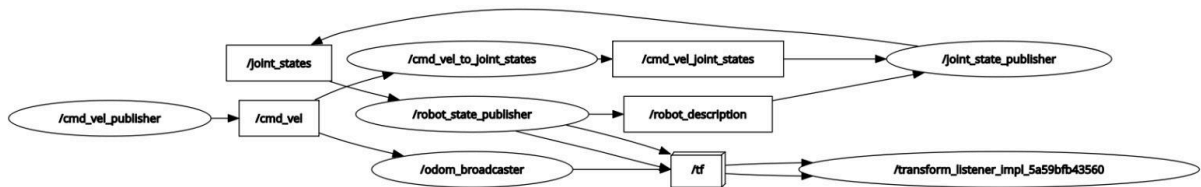


Figure 13. rqt_graph of `/cmd_vel` implementation including `odom_broadcaster`

[Figure 13](#) shows the message flow where `/cmd_vel_publisher` sends velocity commands to multiple listener (subscriber) nodes simulating movement and odometry feedback in RViz2.

As a main goal, on real hardware, Raspberry Pi 4B running on ROS2 Jazzy, subscribed to /cmd_vel through ROS2 Discovery protocol. You can see the working principle on the following figure ([Figure 14](#)).

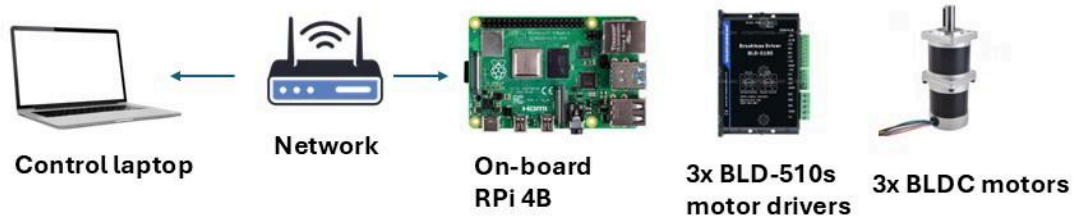


Figure 14. Working principle of Semubot

Python node converted velocity commands to GPIO signals, which made motors running. As a result, there was a successful synchronized movement in RViz2 and the real world ([Figure 15](#)).

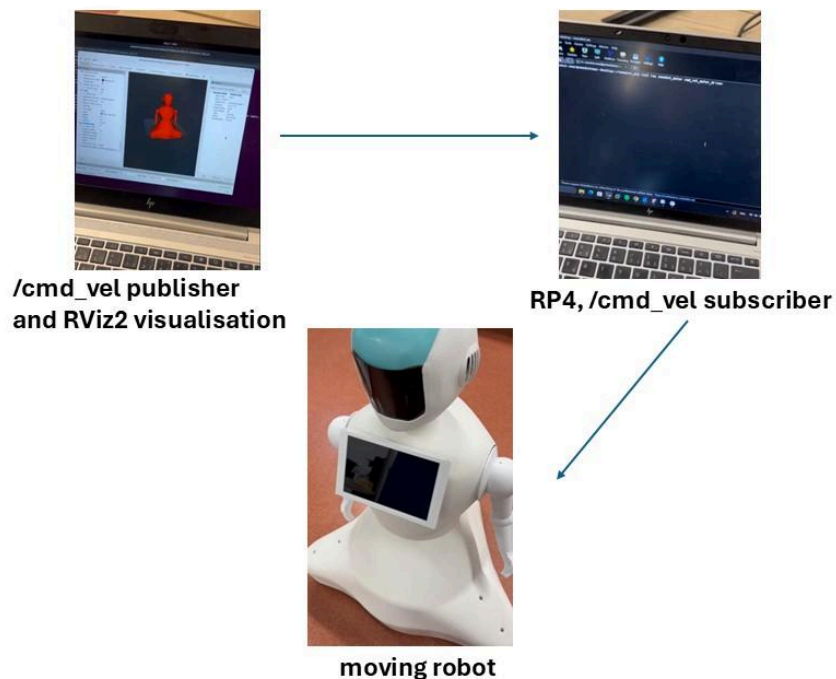


Figure 15. Testing of Semubot with ROS2

Synchronized movement of Semubot in RViz2 and real world can be found in the github repository [\[Appendix 1\]](#).

3.7 Challenges

During development of this project, I encountered several challenges.

One of the challenges was configuring the ROS2 Discovery protocol over a local network between the external laptop and Raspberry Pi 4B so that they can communicate between each other. Although both of them were on the same subnet (same network), ROS2 nodes did not “see” each other. Troubleshooting this issue consisted of repeatedly restarting the ROS2 environment, checking firewall settings, and configuring DDS. After time consuming troubleshooting, this problem was resolved.

Another challenge was constructing the URDF file of Semubot from scratch. Aligning the robot's components in coordinate space required extensive manual adjustments, because STL meshes were created separately for each part. After several iterations, the robot's arms, body, and omniballs were connected to each other.

Integration of Semubot in CoppeliaSim was one of the main challenges. After importing Semubot's URDF file in the simulator, CoppeliaSim displayed a warning that some STL meshes were non-convex and not suitable for simulation. This problem lead to issues with the robot's stability. Even if ROS2 interface plugin was enabled, and `/sim_ros2_interface` (CoppeliaSim's ROS2 node) was active, a custom Lua script attached to robot's `base_link` did not subscribe to `/cmd_vel` topic.

3.8 DISCUSSION

With this project, improvement over the earlier implementation of Semubot's motion control system was achieved. As mentioned before, the previous motion control system included a standalone Python script running on Raspberry Pi. It used joystick input and GPIO-based motor commands, but lacked modularity and integration with robotic simulation and visualization tools.

ROS2-based framework represents flexible middleware, enabling communication between nodes. Moreover, it gives the opportunity to visualize robot in RViz2 and CoppeliaSim simulators. This middleware will enable such capabilities as camera-based perception, voice interaction, without rewriting core communication logic. The integration with the CoppeliaSim simulator also gives the opportunity to test new features in a safe virtual environment, before deploying to Semubot itself. Therefore, this work not only improves the control system of Semubot but also gives solid foundation for more advanced behaviours for Semubot in future.

In conclusion, by transitioning to the ROS2 framework, Semubot approaches becoming a fully modular humanoid robot.

SUMMARY AND FUTURE WORK

This thesis presented the development of ROS2 middleware for Semubot, the first open-sourced humanoid robot in Estonia. This work included modelling of Semubot's physical structure using URDF, implementing real-time velocity control through `/cmd_vel` topic. After, there was integration motor control on real hardware, Raspberry Pi 4B.

Successful synchronization between ROS2 simulation in RViz2 and real-world robot movement was achieved. Additionally, a Digital Twin of Semubot was created in CoppeliaSim to enable virtual testing with world-like physical properties.

Future Work

Future development of Semubot can focus on several areas. Currently, motor control is achieved through Raspberry Pi 4B. However, switching to microcontroller such as STM32 can provide increased flexibility, modularity, and reduced latency, because STM32 microcontroller operate without any embedded operating system. Moreover, improving STL meshes and Lua would lead to correct subscription to ROS2 topic and simulation in CoppeliaSim.

Functionality of Semubot can be expanded by integrating autonomous navigation, advanced perception through camera and lidar sensors, and voice interaction for being full humanoid robot.

Acknowledgements

I would like to extend my thanks to the University of Tartu's Science and Technology program for providing me with great education and opportunities for both my career and education growth. I am especially grateful to Assoc. Prof. Ilona Faustova, the Vice Director of Institute of Bioengineering.

Moreover, I wish to express gratitude to my thesis supervisor, Renno Raudmäe.

I am profoundly thankful to my family members for their support during my studies.

Thank you all for your contributions.

Appendices

Appendix 1. Github repository

<https://github.com/SemuBot/Huseynli-thesis-2025-ROS-integration>

REFERENCES

- [1] Asamaka Industries Ltd. (2025, January 3). Exploring the impact of humanoid robots on enhancing social and learning skills in children with autism. Association for Advancing Automation. <https://www.automate.org/news/exploring-the-impact-of-humanoid-robots-on-enhancing-social-and-learning-skills-in-children-with-autism>
- [2] Kushniruk, A. W., Nair, H. S., & Borycki, E. M. (2025). The Pepper Robot in Healthcare: A Scoping Review. *Envisioning the Future of Health Informatics and Digital Health*, 101-105. <https://doi.org/10.3233/SHTI250057>
- [3] De Carolis, B., Carofiglio, V., Grimandli, I., Macchiarulo, N., Palestra, G., & Pino, O. (2020, June). Using the pepper robot in cognitive stimulation therapy for people with mild cognitive impairment and mild dementia. In *Proceedings of the ACHI-The Thirteenth International Conference on Advances in Computer-Human Interactions* (pp. 452-457). https://www.thinkmind.org/index.php?view=article&articleid=achi_2020_5_540_20203
- [4] Bertacchini, F., Demarco, F., Scuro, C., Pantano, P., & Bilotta, E. (2023). A social robot connected with chatGPT to improve cognitive functioning in ASD subjects. *Frontiers in Psychology*, 14, 1232177. <https://doi.org/10.3389/fpsyg.2023.1232177>
- [5] Getson, C., & Nejat, G. (2023). Human-robot interactions with an autonomous health screening robot in long-term care settings. *Advanced Robotics*, 37(24), 1576-1590. <https://doi.org/10.1080/01691864.2023.2293133>
- [6] Zarubica, S., & Bendel, O. (2023, December). Pepper as a Learning Partner in a Children's Hospital. In *International Conference on Social Robotics* (pp. 15-26). Singapore: Springer Nature Singapore. https://doi.org/10.1007/978-981-99-8718-4_2
- [7] IEEE Spectrum. (n.d.). *Pepper*. ROBOTS: Your Guide to the World of Robotics. <https://robotsguide.com/robots/pepper>
- [8] Pino, O., Palestra, G., Trevino, R., & De Carolis, B. (2020). The humanoid robot NAO as trainer in a memory program for elderly people with mild cognitive impairment. *International Journal of Social Robotics*, 12, 21-33. <https://doi.org/10.1007/s12369-019-00533-y>
- [9] 3WAY d.o.o. (n.d.). NAO 6 robot. <https://www.3way-world.com/izdelek/nao-6-robot/>
- [10] Bakken, D. E. (2001). *Middleware*. Washington State University. <https://eecs.wsu.edu/~bakken/middleware-bakken.pdf>

- [11] Elkady, A., & Sobh, T. (2012). Robotics Middleware: A Comprehensive Literature Survey and Attribute-Based Bibliography. *Journal of Robotics*, 2012(1), 959013. <https://doi.org/10.1155/2012/959013>
- [12] Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., ... & Ng, A. Y. (2009, May). ROS: an open-source Robot Operating System. In *ICRA workshop on open source software* (Vol. 3, No. 3.2, p. 5).
- [13] Nagy, E. (2024). Processing sensor data from the NAO robot. *Journal of Applied Technical and Educational Sciences*, 14(4), 392-392. <https://doi.org/10.24368/jates392>
- [14] MuleSoft. (n.d.). What is an API? (Application Programming Interface). MuleSoft. <https://www.mulesoft.com/api/what-is-an-api>
- [15] Perera, V., Pereira, T., Connell, J., & Veloso, M. (2017). Setting up pepper for autonomous navigation and personalized interaction with users. *arXiv preprint arXiv:1704.04797*. <https://doi.org/10.48550/arXiv.1704.04797>
- [16] Wikipedia contributors. (2025, April 18). *Simultaneous localization and mapping*. Wikipedia. https://en.wikipedia.org/wiki/Simultaneous_localization_and_mapping
- [17] Fitzpatrick, P. M., Natale, L., & Metta, G. (2010, January). YARP, a Thin Middleware for (Humanoid) Robots. In *Enabling Intelligence through Middleware*.
- [18] Wikipedia contributors. (2025, May 19). *Interface description language*. Wikipedia. https://en.wikipedia.org/wiki/Interface_description_language
- [19] Wikipedia contributors. (2025, May 20). SWIG. Wikipedia. <https://en.wikipedia.org/wiki/SWIG>
- [20] Metta, G., Fitzpatrick, P., & Natale, L. (2006). YARP: yet another robot platform. *International Journal of Advanced Robotic Systems*, 3(1), 8. <https://doi.org/10.5772/5761>
- [21] Ng-Thow-Hing, V., List, T., Thórisson, K. R., Lim, J., & Wormer, J. (2007, October). Design and evaluation of communication middleware in a distributed humanoid robot architecture. In *IROS'07 Workshop Measures and Procedures for the Evaluation of Robot Architectures and Middleware* (Vol. 29).
- [22] Macenski, S., Foote, T., Gerkey, B., Lalancette, C., & Woodall, W. (2022). Robot operating system 2: Design, architecture, and uses in the wild. *Science robotics*, 7(66), eabm6074. <https://doi.org/10.1126/scirobotics.abm6074>

- [23] Woodall, W. (2014, June). *ROS on DDS*. ROS 2 Design https://design.ros2.org/articles/ros_on_dds.html
- [24] Fazzari, K. (2019, July). *ROS 2 DDS-Security integration*. ROS 2 Design. https://design.ros2.org/articles/ros2_dds_security.html
- [25] Bono, A., Brameld, K., D'Alfonso, L., & Fedele, G. (2024, August). Open Access NAO (OAN): a ROS2-based software framework for HRI applications with the NAO robot. In 2024 33rd IEEE International Conference on Robot and Human Interactive Communication (ROMAN) (pp. 522-527). IEEE. <https://doi.org/10.1109/RO-MAN60168.2024.10731252>
- [26] Jiang, P., Ergu, D., Liu, F., Cai, Y., & Ma, B. (2022). A Review of Yolo algorithm developments. *Procedia computer science*, 199, 1066-1073. <https://doi.org/10.1016/j.procs.2022.01.135>
- [27] Montenegro, G., Chacón, R., Fabregas, E., Garcia, G., Schröder, K., Marroquín, A., ... & Farias, G. (2022). Modeling and control of a spherical robot in the CoppeliaSim simulator. *Sensors*, 22(16), 6020. <https://doi.org/10.3390/s22166020>
- [28] Michel, O. (1998). Webots: Symbiosis between virtual and real mobile robots. In *Virtual Worlds: First International Conference, VW'98 Paris, France, July 1-3, 1998 Proceedings 1* (pp. 254-263). Springer Berlin Heidelberg. https://doi.org/10.1007/3-540-68686-X_24
- [29] Rohmer, E., Singh, S. P., & Freese, M. (2013, November). V-REP: A versatile and scalable robot simulation framework. In *2013 IEEE/RSJ international conference on intelligent robots and systems* (pp. 1321-1326). IEEE. <https://doi.org/10.1109/IROS.2013.6696520>
- [30] Koenig, N., & Howard, A. (2004, September). Design and use paradigms for gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ international conference on intelligent robots and systems (IROS)(IEEE Cat. No. 04CH37566)* (Vol. 3, pp. 2149-2154). Ieee. <https://doi.org/10.1109/IROS.2004.1389727>
- [31] Ayala, A., Cruz, F., Campos, D., Rubio, R., Fernandes, B., & Dazeley, R. (2020, October). A comparison of humanoid robot simulators: A quantitative approach. In 2020 Joint IEEE 10th international conference on development and learning and epigenetic robotics (ICDL-EpiRob) (pp. 1-6). IEEE. <https://doi.org/10.1109/ICDL-EpiRob48136.2020.9278116>
- [32] Andreiev, A., & Sotnik, S. (2024). Comparative analysis of robotics platform: Webots, Coppeliasim and Gazebo.
- [33] Open Robotics. (n.d.). urdf. ROS Wiki. <https://wiki.ros.org/urdf>

- [34] Open Robotics. (n.d.). RViz user guide. ROS 2 Documentation: Jazzy. <https://docs.ros.org/en/jazzy/Tutorials/Intermediate/RViz/RViz-User-Guide/RViz-User-Guide.html>
- [35] Georgs, N. (2024). *Holonomic motion drive system of a social humanoid robot SemuBot* <https://dspace.ut.ee/items/1fc773b7-1112-4a29-aaef-6a6ecf3e7853>
- [36] Raspberry Pi Ltd. (n.d.). Raspberry Pi 4 Model B specifications. <https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/>
- [37] Oyostepper. (n.d.). BLS-510 brushless digital DC motor driver manual [PDF]. <https://www.oyostepper.com/images/upload/File/BLS-510.pdf>
- [38] Oyostepper. (n.d.). 57BLR70-24-02-HG50 24V 172W 70RPM brushless geared DC motor 50:1 with high precision gearbox BLDC gear motor. Oyostepper. <https://www.oyostepper.com/goods-1145-24V-172W-70RPM-Brushless-Geared-DC-Motor-50-1-with-Gearbox-BLDC-Gear-Motor.html>

Non-exclusive licence to reproduce thesis and make thesis public

I, Omar Huseynli

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,

“ROS Integration for the Semubot Robot”

supervised by Renno Raudmäe

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.
3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.
4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Omar Huseynli

20.05.2025