Zurich University of Applied Sciences

Department Life Sciences and Facility Management

Institute of Computational Life Sciences

PROJECT WORK 2

Digital Twin for Autonomous Robots using ROS2

Author:
Noirin Graham

Supervisors:
Dr. Christian Glahn
Christoph Koller

Submitted on
December 11, 2024

Study program:
Applied Digital Life Sciences, B.Sc.

Imprint

Project: Project Work 2
Title: Digital Twin for Autonomous Robots using ROS2
Author: Noirin Graham
Date: December 11, 2024
Keywords: robotics, gazebo, rviz, ubuntu, linux, ros2 rolling, digital twin,AMRs, URDF, SDF, Robot State Publisher

Study program:
Applied Digital Life Sciences, B.Sc.
Zurich University of Applied Sciences

Supervisor 1:
Dr. Christian Glahn
Zurich University of Applied Sciences
Email: glah@zhaw.ch
Web: Link

Supervisor 2:
Christoph Koller
Zurich University of Applied Sciences
Email: kolc@zhaw.ch
Web: Link

Declaration of Authorship

REMOVE THIS SECTION IF THE ORIGINAL COPY OF THE ZHAW DECLARATION OF ORIGINALITY IS USED IN THE APPENDIX.

I, Noirin Graham, declare that this thesis titled, "Digital Twin for Autonomous Robots using ROS2" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at the Zurich University of Applied Sciences.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this university or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

Abstract

This project presents the development of a simplified digital twin of an autonomous robot. The threewheeled robot was recreated as a digital twin using ROS2, an advanced open-source robotics software. The project was developed in the ros2 rolling distribution. Throughout the project, the robot's description could be successfully captured in a developed URDF file, which used the marco language xacro. The URDF file could be spawned successfully in the visualization tool rviz2 as well as the simulator gazebo ionic. For the control system within the gazebo environment, the differential diver plugin was implemented. The two repositories ros_arduino_bridge and serial_motor_demo were used as a starting point for the robot control system. These repositories contained a network of ros2 nodes, which communicated motor speed over the motor_command topic. To tailor the given nodes to the requirements of the project an additional node was developed, which communicated with the /cmd_vel and motor_command topics. The main goal of the project could be met, by simply finding some phases of the project. Instead of a sophisticated control system like the ros2_conrtol and ros2_controller packages, the native communication system relying on nodes and topics was used in the project. Further, the connection between the gazebo simulator and the rviz2 visualizer and feedback messages from the motor controller to the rviz2 visualizer could not be established. However, the synchronization of the DT and the realworld robot movement could be established.

Contents

List of Figures

List of Abbreviations

ROS2	**R**obot **O**perating **S**ystem **2**
RViz2	**R**OS **V**isualization tool version **2**
Gazebo	Simulation tool for robotics
DT	**D**igital **T**win
URDF	**U**nified **R**obot **D**escription **F**ormat
Xacro	**X**ML **Macro**
SDF	**S**imulation **D**escription **F**ormat
AMR	**A**utonomous **M**obile **R**obot
NIST	**N**ational **I**nstitute of **S**tandards and **T**echnology
PWM	**P**ulse **W**idth **M**odulation
LTS	**L**ong-**T**erm **S**upport
TF	**T**ransform (in ROS for coordinate transformations)
tf2	ROS **T**ransform library version **2**
Odom	**O**dometry (for estimating position and velocity)
ROS_DOMAIN_ID	ROS domain configuration parameter
ROS_IP	IP address configuration in ROS
cmd_vel	**C**ommand **Vel**ocity (topic in ROS for movement control)
gz_ros_bridge	**G**azebo-**R**OS **bridge**
Colcon	Build tool for ROS2 workspaces
CMake	Cross-platform build system
XML	**E**xtensible **M**arkup **L**anguage
YAML	**YAML** Ain't **M**arkup **L**anguage (used for configuration files)
Raspberry Pi	Single-board computer used as an onboard device
Arduino	Microcontroller board
SSH	**S**ecure **Sh**ell (for secure remote communication)
IDE	**I**ntegrated **D**evelopment **E**nvironment
WSL2	**W**indows **S**ubsystem for **L**inux **2**
VirtualBox	Virtualization software for running virtual machines
Docker	Platform for containerized applications
Differential Drive Plugin	Gazebo plugin for differential drive robot control
Rolling Distribution	Development release of ROS2 with latest features
Jazzy Jalisco	Latest LTS distribution of ROS2
Harmonic/Ionic	Gazebo distribution names
Motor Driver	Interface converting control signals for motor operation
Motor Controller	Device managing motor speeds and directions
Robot State Publisher	ROS node for publishing robot states
Joint State Publisher	ROS node for publishing joint states
TF Tree	Hierarchical structure of coordinate transformations in ROS
ROS Node	Independent computing process in ROS
ROS Topic	Communication channel for ROS messages
Rolling Ridley	Developer release version of ROS2 distribution

Chapter 1

Introduction

As urbanization continues to expand, integrating sustainable green spaces into densely populated areas has become a growing challenge. Rooftop solar panels, already serving as key assets in renewable energy production, present an opportunity to co-function as natural green spaces, contributing to urban biodiversity and improved air quality. However, maintaining vegetation in these environments is an exhausting task, as overgrown plants can cast shadows on photovoltaic panels, reducing their efficiency but the human interference within the green spaces preferentially is kept to a minimum.

To address this an autonomous robot capable of maintaining rooftop vegetation will be developed. The robot will be equipped with advanced decision-making capabilities to identify and trim specific plants without hindering the photovoltaic panels performance. To ensure optimal functionality and provide real-time monitoring, a digital twin will be developed to simulate, control, and oversee the robot's actions. This innovative solution combines robotics, artificial intelligence, and digital simulation to create a sustainable coexistence between green infrastructure and renewable energy technology.

Controlling robots is complex and requires a lot of knowledge and experience, among the systems that reduce the complexity of controlling robots is the Robot Operating System (ROS2). Therefore, as a starting point, this project aims to learn and understand ROS2 and its applications in a concrete robotic application. Therefore, a simplified version of the envisioned autonomous robot will be used in this project.

1.1 Project aim

Within the ROS2 ecosystem, a digital twin will be developed, focusing on a control system managing the DT and the robot. ROS2 defines the components, interfaces and tools for building advanced robots. A robot is built out of three primary elements: actuators, sensors, and a control system. Actuators enable the robot to perform movements, sensors allow it to perceive its environment, and the control system functions as the robot's brain, processing inputs and coordinating actions. ROS2 enables users to build these components and create a connection between them using ROS2 tools, which are called topics and messages. The visualization and simulation environment, such as rviz2 and gazebo, provides powerful tools for developing and testing robotic systems in a virtual space before deploying them in the real world (Open Robotics, n.d.-e).

Chapter 2

Literature Review

This literature review examines the background and relevant technologies that form the foundation for the project, focusing on ROS2, its components, and the role of digital twins in robotic applications.

2.1 Digital Twin

Digital Twin (DT) technology is an emerging field that continues to expand rapidly. Although many industries have begun adopting this technology, the definition of a Digital Twin remains ambiguous in the literature. Some authors define it as a virtual representation of a physical system, while others emphasize the exchange of information between two domains, incorporating sensors, data, and models (Stączek et al., 2021). In this report, the definition provided by the National Institute of Standards and Technology (NIST) will be used:

> "A digital twin is the electronic representation—the digital representation—of a real-world entity, concept, or notion, either physical or perceived (Voas, 2021)."

A DT functions as a model or simulation that offers static or dynamic views of its real-world counterpart. It integrates modeling, simulation, and real-time data to replicate the behaviour of its physical counterpart. Unlike static models, DTs are continuously updated in real-time. The distinction between a DT and a mirroring system often overlaps, making it challenging to define a clear boundary. In this project, a digital twin is defined as a electronic representation of its real-world counterpart, capturing its electrical and functional characteristics. While both components can be controlled simultaneously, their behaviors remain distinct and unique to each system. Conversely, a mirroring system involves two counterparts that move synchronously, with the motion dictated either by the virtual model or the real-world robot. DTs are applicable in various domains, such as simulating behaviour over time (Dynamic Modeling) or controlling physical systems through bidirectional communication, such as managing unmanned aerial vehicles (UAVs) or autonomous mobile robots (AMRs). However, this technology introduces several challenges, including cybersecurity risks, heightened complexity, and ensuring the accuracy of the digital twin representation (Voas, 2021).

2.2 Autonomous Robots

An autonomous robot is defined as a system that can handle various environmental constraints automatically while completing a task. It is equipped with the necessary methods to adjust to its surroundings without requiring constant human input (Wahde, 2016). The predecessors of autonomous robots are conventional robots that are widely used in manufacturing across various industries for over two decades. Conventional robots operate by executing preprogrammed instructions or direct commands. Autonomous robots are designed to increase flexibility and adaptability. These advanced systems can adapt to environmental constraints and perform tasks with minimal human intervention. A notable subclass of autonomous robots is mobile robots (Fahimi, 2009). In this project, the focus will lay on developing a digital twin of an autonomous mobile robot (AMR). AMRs, the latest advancement in unmanned mobile robots, are predominantly used in large warehouses for material handling. These robots are designed to move across a defined area and navigate through an environment to perform a specific task, while human intervention is kept at a minimum. Decisions on where and when to act are based on predefined programming and environmental inputs. In order to develop an AMR that is able to perform tasks autonomously a DT is essential. The open-source software ros2 offers a variety of a highly functional tool, which simplifies the process of building a DT significantly (Stączek et al., 2021).

2.3 What is ROS?

The robot operating system is an advanced open-source robotics software that provides a collection of software libraries and tools to help developers build robot software, offering features such as hardware abstraction, device drivers, libraries for commonly used algorithms, and message-passing between processes. The first version of ROS had a significant role in the maturing robot industry, the ROS ecosystem has become influential in nearly every intelligent machine sector. The successor ROS2 was developed to address the limitations such as lack of security, low reliability in non-ideal environments and limited real-time support of its predecessor. The architecture of ROS2 is divided into three categories Middleware, Algorithms and Developer tools (Macenski et al., 2022).

2.3.1 What is a ROS distribution

A distribution contains a collection of interrelated software components, tools, libraries and infrastructure that are bundled together and released at specific intervals. ROS as well as ROS2 are available in a sequence of distributions, these are essentially stable releases that include the core components which are required to build and run robotic systems. In the ROS ecosystem, there are two main types of distributions available: Long-Term Support (LTS) distributions and non-LTS (development) distributions. The main difference between the two lies in their update and maintenance policies. LTS distributions are designed for stability and reliability, with updates provided at defined intervals, primarily focusing on bug fixes and security patches. The latest long-term support(LTS) distribution is the "Jazzy Jalisco", which was released on May 23rd, 2024 (Open Robotics, n.d.-d).

In contrast, non-LTS (development) distributions reflect the latest state of ROS development. This distribution includes the latest changes and features of ROS 2. Unlike other stable distributions, the non-LTS distribution is not focused on long-term stability but instead is constantly updated. It serves two purposes:

1. It is a staging area for future stable distributions of ROS 2, and
2. It is a collection of the most recent development releases.
3. allows for testing existing solutions for upcoming ROS distributions.

Due to it's frequent changes and potential instability it is not recommended for use in production environments. Within the ROS ecosystem, the rolling distribution takes on the role of the non-LTS (development) distributions (Open Robotics, n.d.-b).

2.3.2 Nodes and Topics

The simple explanation of a node is the following; nodes are executable files within a ROS2 package. Each node should be responsible for a single modular purpose. To send and receive data from other nodes, messages can be sent to a topic (Open Robotics, n.d.-f). A graphical overview of the interaction between nodes and topics is given in the figure 2.1. In order for this system to work properly the two nodes need to be accessible from the same network, further information regarding the network configuration will be discussed in the next chapter .

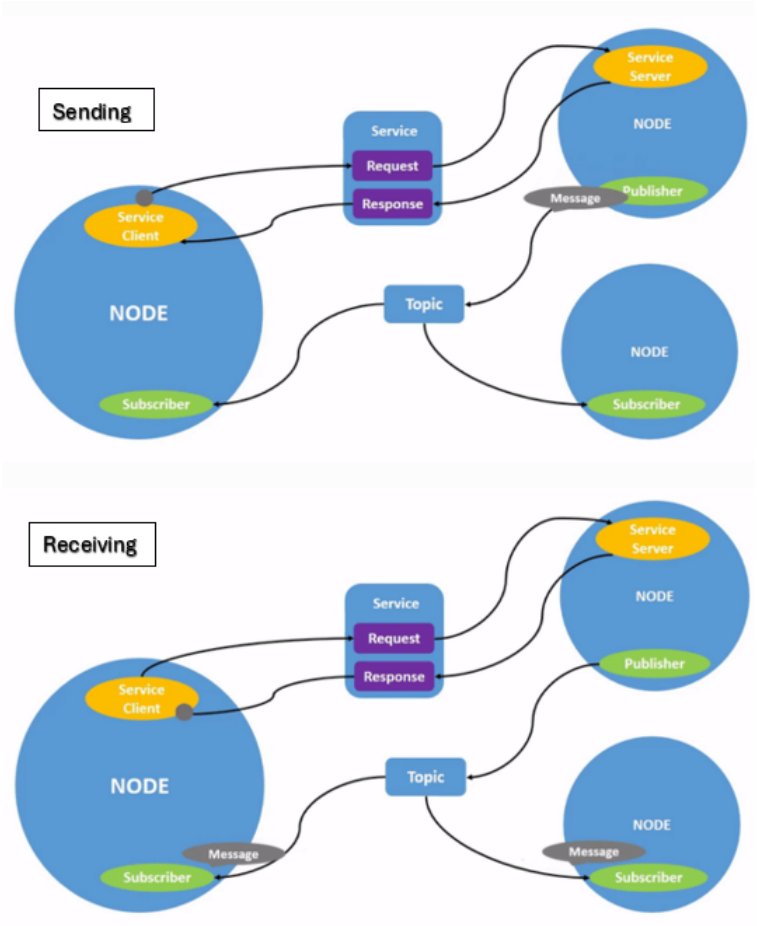

FIGURE 2.1: URDF components relationships
(**Interaction nodes and topics**)

2.4 Rviz and Gezebo

ROS2 provides a variety of visualization and simulation tools, some developed within the ROS ecosystem and others designed to be compatible with it. Within the scope of this Project, the visualization tool RViz and the simulation tool Gazebo were chosen to support the development and implementation of the digital twin.

2.4.1 RViz

RViz is a ROS toolkit that enables the visualization of data and algorithms in real-world domains, including 2D and 3D spaces. It can work with various data structures, enabling its use across different computational and scientific fields. (Kam et al., 2015)

2.4.2 Gazebo

Gazebo is an open-source collection of modular software libraries aimed at simplifying the development of high-performance robotics and simulation applications.(Open Robotics, n.d.-a)

2.4.3 Combination of RViz and Gazebo

RViz itself does not provide a built-in simulation, therefore a parallel usage of Gazebo is widely used. Gazebo handles the physics-based simulation of robots and their environments, while RViz provides a real-time visual interface to display the simulation data. Together, they allow users to visualize the robot's movements, map generation, sensor data, and coordinate transformations, as well as to interact with the robot by sending commands or testing navigation and planning strategies in a controlled environment.(Indri, n.d.) In order to develop and implement a digital twin with the support of the two tools the robot's physical structure and configuration must be defined in a specific format.

2.5 URDF, Xacro and SDF

The Unified Robotic Description Format (URDF) is an XML file format used in ROS and ROS2 to define a robot's physical structure and configuration. It provides a 3D representation of the robot, specifying details about its joints, links, sensors, and their respective properties. This effectively describes the robot's physical design and behaviour. The diagram, as shown in Figure 2.2, form the ROS wiki illustrates the relationships between these components (Open Robotics, n.d.-g). URDF files are created to define the kinematic and dynamic properties of a single robot in isolation. To develop a more complex system, which includes sensors, actuators or a second robot most developers use the macro language xacro (XML macro). It enhances URDF files by allowing for conditional logic, variables, and parameterization, enabling developers to make these files configurable and maintainable. A Modular Xacro splits up the robot description into multiple separate child Xacro files, each with a specific purpose. These files are then included in a parent Xacro file. The modular structure typically includes:

1. Robot Description File: Contains the main robot model description, such as links, joints, and kinematics.

2. Material List File: Defines materials and visual properties, such as colors and textures used in the model.

3. Configuration File: Includes parameters and settings required for simulation in tools like Gazebo, such as physical properties and plugin configurations.

A Xacro file itself is an enhanced version of a URDF (Unified Robot Description Format) file, enriched with Xacro-specific tags to support macros, parameterization, and modular inclusion. At runtime, the Xacro program preprocesses the parent Xacro file along with its included files to generate a standard URDF. This URDF file is then used by simulation tools (e.g., Gazebo) or visualization tools (e.g., RViz) (Albergo et al., 2022). Another way to make URDF suitable for a gazebo simulation, without using Xacro, is to translate the URDF file to the native Simulation Description Format (SDF) of Gazbeo. The translation process includes the implementation of simulation-specific tags to the URDF file. These tags define attributes such as the robot's pose, friction, inertial elements, and other physical properties. Furthermore, these tags can describe the surroundings of the robot and thereby place the robot in a simulated world. Converting a URDF to SDF is straightforward and is achieved by adding Gazebo plugins to the URDF file. Essentially, Gazebo plugins create a

communication interface (referred to as a Topic in ROS) between ROS and Gazebo. The control processes in ROS communicate through a Publish/Subscribe model on these Topics (Takaya et al., 2016).

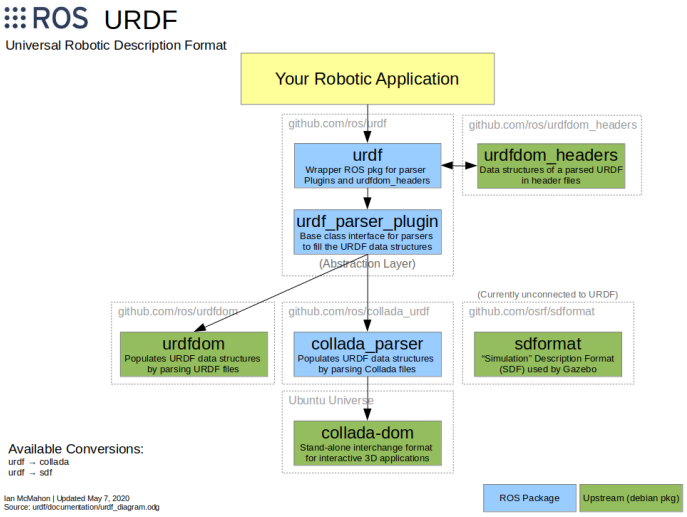

FIGURE 2.2: URDF components relationships

(Open Robotics, n.d.-g)

Chapter 3

Methods

3.1 Project Pipline

The project pipeline contains 8 distinct phases that are shown in the figure 3.1. The pipeline begins by building the physical prototype of the robot. The second phase involved setting up the hardware and software required for the project, which included the initial introduction to the ROS 2 ecosystem. This phase involved learning and implementing foundational concepts necessary for the subsequent stages of the project. The third phase implements the ROS2 installations and tests its functionality by establishing a connection through two nodes, one running on the onboard computer and another on the control unit. The fourth phase carries out the development of the URDF file. The fifth phase vizualizes the URDF file in the RViz2 visualizer, and establishes the joint movement by using the package "joint state publisher." The sixth phase includes the simulation of the model within the Gazebo simulator. The DiffDrive plugin was used to control the simulation. The seventh phase tested the remote control of the motor of the physical robot by using two ROS2 nodes: one listening for motor speed commands and one sending the motor speed commands. In the last phase, the three components—RViz, Gazebo, and drive node—were concatenated in a launch file and run simultaneously. This pipeline doesn't necessarily follow the structure of a best practice case, it shows how this project was built.

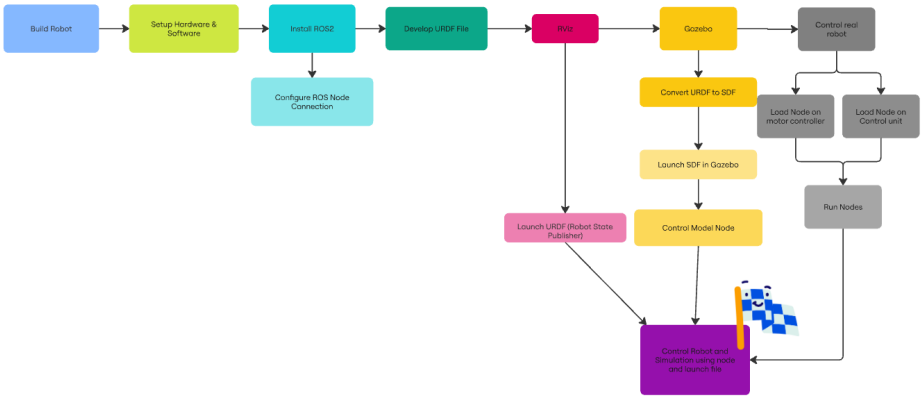

FIGURE 3.1: Project Pipline

(Open Robotics, n.d.-g)

3.2 Robot parts

The core elements may differ for different robot types, for this project, the core elements which were used to control the robot were the following:

1. motors
2. power stations
3. the motor drive
4. the motor drive
5. onboard computer

The motors are the basic components which were used to give the robot the ability to move. In order to run the motors a power station which provides the appropriate voltage was included. By supplying the motors with electricity a basic movement of the motor could be achieved. To change the speed or direction and control them remotely, the robot was utilised with a motor driver. The motor driver took a low voltage and low current signal, primarily in the form of pulse width modulation (PWM), from the controller and amplified it using the power supply, creating a higher voltage and current feed to drive the motor. The PWM signal consisted of a series of fast pulses, where the averaged value over time determined the percentage of the total voltage supplied to the motor. This ratio of on-time to off-time, known as the duty cycle, was implemented to regulate motor operation. The motor control used more practical input formats, such as target speed, to calculate the appropriate signal for the driver. The driver, in turn, amplified this signal and transmitted it to the motor. A communication layer, a serial connection, was established to link the onboard computer with the motor controller. This communication allowed the onboard computer to send target speeds to the motor control. The onboard computer was equipped with ROS2. The key tasks of ROS2 were to calculate the target motor speeds and translate these into a protocol compatible with the motor controller.

3.3 System Setup of computer hardware and Software

There are various computer hardware and software configurations available for working with ROS2, ranging from using a virtual machine with a Linux Ubuntu operation system (OS), to running a Docker container with a selected ROS2 image that can be run on a Windows OS by using the WSL2 (Docker, n.d.), or directly installing ROS2 on computer hardware that has Ubuntu Linux installed. In this project, all the previously mentioned methods were explored. The detailed experience will be discussed in more detail in the results in the chapter ”Installation computer hardware and software”.

3.3.1 ROS2 installation

The next step included the installation of the ROS2 Rolling on both the control unit and edge device. A detailed description of the installation can be found the installation guide A2. For Ubuntu 24.04 LTS the ROS2 distribution Rolling Ridely or Jazzy Jalisco can be used. For this stage of the project, the Rolling Ridely distribution was used. The Rolling Ridely is the rolling development distribution,

which means it continuously integrates the latest features, updates and improvements (Open Robotics, n.d.-c). This also makes it more unstable than the Jazzy Jalisco, which introduces several complications, as will be discussed in a later chapter. To guarantee a stable model in the future, a distribution change will be necessary.

3.3.2 Network configuration

The network structure can be set up in multiple ways and should be dependent on the requirements a project has. For this project, two existing networks—home and ZHAW—were planted to be utilized.

3.3.3 ROS2 Nodes

The development of the project relied heavily on the usage and development of Nodes and the communication system between nodes and topics. The primary used node was the demo node talker/listener, it was used to test the communication between the control unit and the edge device. These six additional nodes were used within this project:

- `robot_state_publisher`: Publishes the URDF model to the /robot_description topic for visualization and simulation.
- `joint_state_publisher`: Publishes the robot's joint states for visualizing movements in RViz2.
- `rviz2`: Launches RViz2 with a predefined configuration to visualize the robot's state.
- `ros_gz_sim`: Simulates the environment and robot model based on control commands.
- `ros_gz_bridge`: Connects ROS 2 and Gazebo to enable communication for movement commands.
- `cmd_vel_to_motor_command`: Converts /cmd_vel commands into motor commands for the real robot and Gazebo.

In order for these six nodes to send and receive data from each other the following topics where used:

- `/robot_description`: Publishes the robot's URDF model for visualization in RViz2 and simulation in Gazebo.
- `/joint_states`: Publishes the state of the robot's joints.
- `/cmd_vel`: Sends velocity commands (linear and angular) to control the robot's movement.
- `/motor_command`: Simulates the environment and robot model based on control commands.
- `ros_gz_bridge`: Transmits motor-specific commands to the real robot's motor controller.

- `/tf`: Publishes transformations for coordinate frames.

3.4 Develop URDF File

To create a digital twin, a robot description is required. This description contains all the information about the robot's physical characteristics and is stored in a URDF file. The structure of a URDF file follows a simple and repetitive pattern. At its core, it consists of a tree of links connected by joints. The links represent the robot's physical components, such as wheels, while the joints define the dynamic relationships between these components—for example, the rotation of a wheel relative to the robot's body. When defining a joint, a specific type must be assigned. The available joint types are:

- Revolute: Rotational motion with defined minimum and maximum angle limits.
- Continuous: Unlimited rotational motion, such as that of a wheel.
- Prismatic: Linear sliding motion with specified minimum and maximum position limits.
- Fixed: A rigid connection where the child link is immovably attached to the parent link, often used for "convenience" links.

(Newans, n.d.)

URDF files are written in XML, with their content represented as a series of nested tags. The primary tags used in this project include were robot tag, link tags and joint tags.

3.4.1 The Robot tag

The robot tag is the root element of the robot description file, encapsulating all other elements. It is placed immediately following the XML declaration. It is placed immediately following the XML declaration. ("Urdf/XML - ROS Wiki", n.d.)

```
<!-- XML declaration -->
<?xml version="1.0"?>
<!-- Robot tag-->
<robot name="batmobil">

   <!--robot links and joints and more-->
   <link>
   </link>

   <joint>
   </joint>

</robot>
```

LISTING 3.1: Robot Tag code Example

3.4.2 Link tags

The link tag defines a rigid body, specifying its visual features, and collision properties. ("Urdf/XML - ROS Wiki", n.d.)

```
<!-- Right Wheel Link -->
 <link name="right_wheel">

 <!-- The visual properties of the link -->
   <visual>
     <geometry>
       <cylinder length="0.24" radius="0.3"/>
     </geometry>
     <material name="yellow" />
   </visual>

 <!-- The collision properties of a link-->
   <collision>
     <geometry>
       <cylinder length="0.24" radius="0.3"/>
     </geometry>
   </collision>

 </link>
```

LISTING 3.2: Link Tag code Example

3.4.3 Joint tags

The joint tags describes the kinematics and dynamics of the joint, including its savety limits. It allways stands between a child link and it's parent link.("Urdf/XML - ROS Wiki", n.d.)

```
 <!-- Joint chassis-->
 <joint name="chassis_joint" type="fixed">

   <parent link="base_link"/>
   <child link="chassis"/>

   <origin xyz="-1 0 0"/>

 </joint>
```

LISTING 3.3: Joint Tag Code Example

Additional tags such as sensor, model state or gazebo can be implementet. In this project, the URDF was used to establish a foundational understanding of how a robot description is configured. For the final simulation in Gazebo, the URDF file was converted into an SDF file, which is compatible with Gazebo and allows the model to be launched in the simulation environment.

3.5 Launch URDF in RViz

To visualize the URDF model, the ROS 2 visualization tool Rviz was used. There are several ways to launch a URDF in Rviz, but the simplest approach is to create a ROS 2 launch file. A ROS 2 launch file allows the simultaneous configuration and execution of multiple ROS 2 nodes. These launch files can be written in Python, XML, or YAML, depending on the project requirements. (**CreatingLaunchFile**) For this project, a Python launch script was created. This script starts the following nodes.

- The robot state publisher node, which publishes the robot's state to the tf2 transform tree,

- The joint state publisher gui node, enabling interactive control to move the robot's wheels, and
- The Rviz node, to visualize the URDF model in the Rviz interface.

```
from launch import LaunchDescription
from launch_ros.actions import Node
from launch.substitutions import Command

def generate_launch_description():
    return LaunchDescription([
        Node(
            package='robot_state_publisher',
            executable='robot_state_publisher',
            output='screen',
            parameters=[{
                'robot_description': Command(['xacro ', '/home/noi/robogardner24/scr/RoboGardener/description/urdf/batmobil.urdf'])
            }]
        ),
        Node(
            package='joint_state_publisher_gui',
            executable='joint_state_publisher_gui',
            name='joint_state_publisher_gui',
            output='screen'
        ),
        Node(
            package='rviz2',
            executable='rviz2',
            name='rviz2',
            output='screen',
            arguments=['-d', '/home/noi/robogardner24/scr/RoboGardener/description/batmobil_config.rviz']
        )
    ])
```

LISTING 3.4: Launch file structure

3.6 Gazebo

The Gazebo simulator was utilized in this project to simulate the robot's behaviour and interactions within a virtual environment. Unlike RViz, Gazebo is not included as a default package in the ROS 2 Rolling installation. Similar to ROS 2, Gazebo is available in multiple distributions, each corresponding to a specific ROS 2 version. For the ROS 2 Rolling distribution, the Gazebo Ionic distribution must be installed. Detailed installation steps for Gazebo in the Rolling distribution are provided in Installation Guide A3. To launch the URDF file two options are given, the primary is to use the xacro program to create the URDF file. By doing so the necessary elements for simulation, such as link inertial properties, joint dynamics, and kinematic chains are included in the URDF file. The second option is to transition the URDF to the native Simulation Description Format (SDF) of gazebo. This option is especially recommended if additional Gazebo-specific features are needed for the model.

3.7 Control the robot

As a starting point to understand how the physical robot could be controlled remotely, a tutorial from the Articubot website was followed. This does not represent

a scientific approach, however, it streamlined this part of the project significantly. The following key steps were recreated based on the tutorial:

- Exploring the setup components and their interactions.
- Establishing an SSH connection from the control unit to the onboard computer using Visual Studio Remote.
- Utilizing the Arduino Extension within the Visual Studio Remote environment to upload scripts to the Arduino Mega (motor controller).
- Transmitting data, such as motor speed, from a node running on the control unit to the appropriate topic via messages, and receiving this data on a corresponding node running on the onboard computer.

Chapter 4

Results

4.1 Repositories for Code Management

A GitHub organization is created to manage all repositories related to the project. Within this organization, four main repositories essential for the project were maintained:

- `digital_twin`: Stored all the code and related documentation for the digital twin model.
- `report`: Contained the technical report of the project.
- `serial_motor`: Held the nodes responsible for executing the model and motor control.
- `ros_arduino_bridge`: Included the Arduino Mega scripts required for the project.

4.2 Build a workspace with colcon

On both machines, a designated workspace is created for the project. To create a workspace within the ROS2 ecosystem the standard build toll colcon must be installed. Colcon simplifies the process of building, testing, and packaging multiple interdependent ROS 2 packages in a single workspace. To create the workspaces the "ROS2 Rolling Beginner: Client libraries tutorial" is followed.(Open Robotics, n.d.-h) Additionally a CMake configuration file and an XML metadata file were generated.

The CMakeLists.txt file defines the build configuration, such as source files, dependencies, and build targets, while the package.xml file provides metadata about the package, including its name, version, authors, and dependencies. Together, these files enable colcon to properly build and integrate the packages into the workspace. (Dirk, n.d.)

4.3 Build of prototype robot

The starting point of the project is the building part of the real-world robot. The initial idea is to create a custom-made robot. This robot is designed with a body measuring 50x50x20 cm and equipped with four wheels, with the two back wheels

motor-controlled. For the body 3D printed material is used. The architecture and control system were inspired by a previous project, in which the control system of the farmbot system is used to control the robot. The robot's case is successfully built, however the connection to the motors from the motor controller where more complicated than predicted.

To streamline the initial stages of the project, the prototype is built using the base of an existing robot system. This recycled robot provided a simple and practical starting point, as it minimized the time and effort required for fabrication and assembly. The design features a three-wheel configuration, with two powered wheels at the back and a single wheel at the front for support and steering. Since this base structure is taking over from an existing robot, the project changed the initial robot design to a three-wheel configuration. The case is made of a transparent plexiglass plate measuring 21.5 x 10.5 x 0.3 cm.

Two power sources, in the form of battery-stacks, are mounted on the case: one dedicated to powering the motors and the other supplying power to the Raspberry Pi. A motordriver is used to convert the high voltage from the power station into a smaller voltage suitable for controlling the motors. This is managed by an Arduino Mega in this project. To conserve energy, an on/off switch is integrate. This is soldered to the existing electrical cables of the motor driver, ensuring a secure and efficient connection. This switch exclusively turns the power station on and off. It extends the existing switch, which manages the flow of electricity between the motor driver and the motor controller. The onboard computer is a Raspberry Pi 4 running the robot control system ROS2 Rolling. To power the Raspberry Pi 4, a Dexter GoPiGo3 head is used as a bridge to connect to the power station. Serial communication over USB connects the onboard computer to the motor controller. The robot's movement is initially established using the FarmBot software.

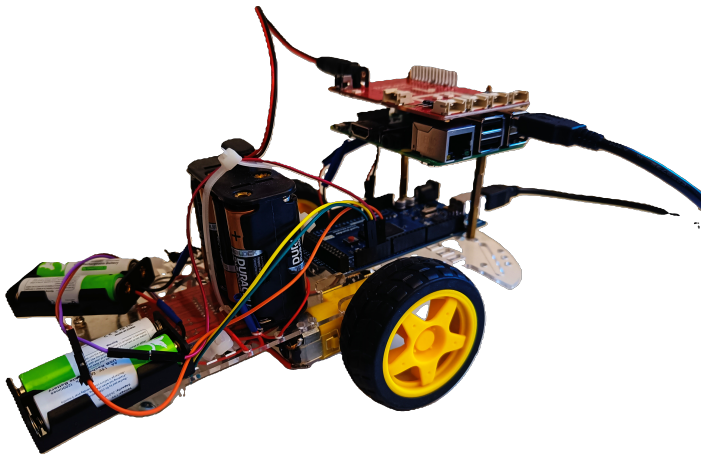

FIGURE 4.1: Prototype "Batmobile"

4.4 Installation hardware and software

The ROS2 ecosystem operates most effectively on a Linux-based system; therefore, a virtual machine with Ubuntu 24.04 is configured using VirtualBox to provide access

to such an environment. This approach is efficient for the first steps of the project. The two tools Rviz2 and Gazebo could be successfully started. Further, the first URDF file of this project is developed within the virtual machine and thereby primarily displayed in the two tools Rviz2 and Gazebo within the virtual machine. The installation of Gazebo initially presented challenges due to incorrect configuration settings, which are explained in detail in the following Gazebo section. The For the further development of the project, the use of the virtual machine created significant challenges, primarily related to establishing a reliable connection between the control unit and the edge device. To address this issue, an attempt is made to connect the control device directly to the network using an Ethernet cable to reduce potential network conflicts. Even though the connection between the two machines could not be established. An alternative approach involved installing a full Ubuntu terminal environment. This Windows-specific solution requires downloading the Ubuntu 24.04 terminal from the Microsoft Store. Windows 10 and 11 include the Windows Subsystem for Linux (WSL2), a compatibility layer designed to run a Linux environment seamlessly within the Windows operating system (Microsoft, 2023). The control machine is connected to the network via an Ethernet cable. While this setup enabled a one-way connection from the control unit to the edge device, the reverse connection remained unsuccessful. The third attempt is to use Docker containers, this approach is a widely used in the industry and has the benefit of running the ecosystem in a separate container. Although a connection between the two containers is successfully established, the installation of Gazebo and the visualization of the robot model failed within this setup. As a result, the control unit is reconfigured with a full installation of the Ubuntu 24.04 operating system, allowing for a fresh and stable environment. The detailed rebooting process can be found in the installation guide A1, placed in the appendix. ROS2 Rolling is successfully installed on both machines, and communication between them is verified using the demo nodes talker and listener. The connection setup involved configuring the ROS DOMAIN ID to 0 and specifying the ROS IP for both machines in their respective .bashrc files. Additionally, the command source "/opt/ros/rolling/setup.bash" is added to the .bashrc files to ensure the ROS setup file is automatically sourced in every new shell session, eliminating the need for manual sourcing.

4.4.1 Network configuration

At home, both devices were successfully connected to the local Wi-Fi, allowing seamless communication between them. However, this setup is not successful in the ZHAW environment. Due to the network configurations at ZHAW, the Raspberry Pi 4 is connected to the IoT network, while the control device is linked to the student network. Communication between these two networks is possible but restricted. To enable communication between two nodes via ROS 2 topics, both nodes must either reside within the same network or be accessible across connected networks. A solution to address the network limitations could not be identified within the available timeframe. Therefore, the ZHAW network is not used within the scope of this project.

4.5 Develop URDF file

The development of the first URDF file (batmobile.urdf) is early established in the project. The guide of the ROS2 Rolling tutorials of the official website and the Articubot one tutorials were followed to develop the URDF file. The batmobile.urdf file is written with the Xacro program, but the structure of the parent file and child file is not used. At the top of the script, the XML version and name of the project were defined. The second section contained information on the material colour. It specified the colour codes which were used to display a specific colour. After the definition of the material colours, the main section started. This section contained links and joints, which describe the robot's visual appearance. Each robot part is represented as a link and each connection between two parts as a joint. In the course of the project, the structure of the URDF file underwent significant modifications. The final URDF file followed the parent-child structure, which is refined based on insights gained through iterative experimentation. The final setup consisted of one parent file and three child files, each serving a specific purpose. The parent file managed the inclusion of child files and organized the overall structure of the robot description. Each child file is included using the <xacro:include> tag. The three child files were the following: `core_model.xacro` : Contained the constants for the robot's dimensions, as well as the joints and links which describe the robot's physics. The file structure of the `diffbot_description.urdf.xacro` from the `ROS2_control_demos` repository is used to build this file (**frohlichROS2_control_demosROS2_c**
`materials.xacro` : Contained the colour codes. `gz_control.xacro`: Contained the Differential Drive Plugin. The Gazebo plugin is used to control and simulate the movement of a robot with a differential drive system. A differential drive system is a type of robot locomotion where two wheels, typically located on either side of the robot, are independently controlled to move the robot forward, backwards, and to rotate around its centre (Rivera et al., 2019).

4.6 Visualization with Rviz2

For the first attempt to launch the model in Rviz2 the shell commands were used:

```
## To run the robot state publisher:
ros2 run robot state publisher robot state publisher --ros-args -p robot
    desription:="$(xacro /path/to/urdf.xacro)"

## To run joint state publisher gui:
ros2 run joint_ state_publisher_gui joint_ state_publisher_gui

## To run rviz2:
rviz2
```

LISTING 4.1: shell commands to run Rviz2

Over the course of the project, a launch file is established to publish the URDF data under the robot description in RViz, further, the GUI State Publisher is enabled, allowing for interactive movement of the robot's wheels. The primary model which is shown in the figure 4.2, contained two wheel arms, which hold the front wheel. This construction is visually closer to the real robot but hinders the free movement of the front wheel in the model. Therefore the second model used a box element to fill the space the arms would be placed in and used a sphere instead of a cylinder element to model the front wheel, as seen in the figure 4.3. The first two models were able to

display the joint movement when using the gui stat publisher panel, but the model's body is static and could not move from one point to another.

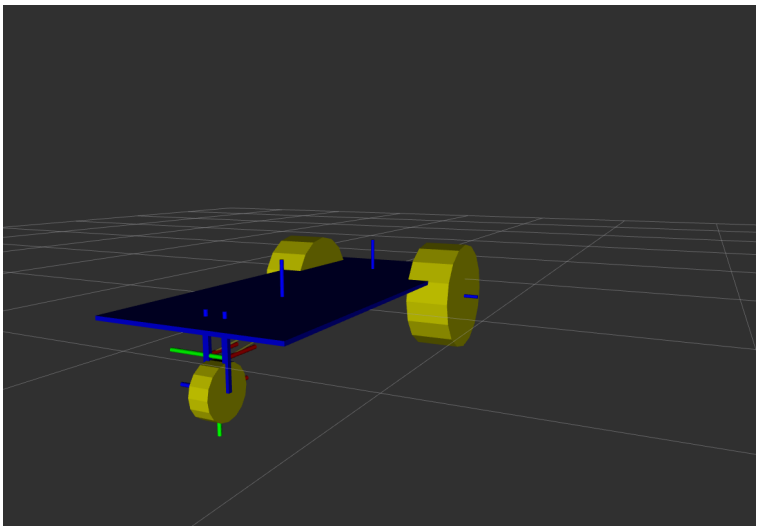

FIGURE 4.2: Primary Model

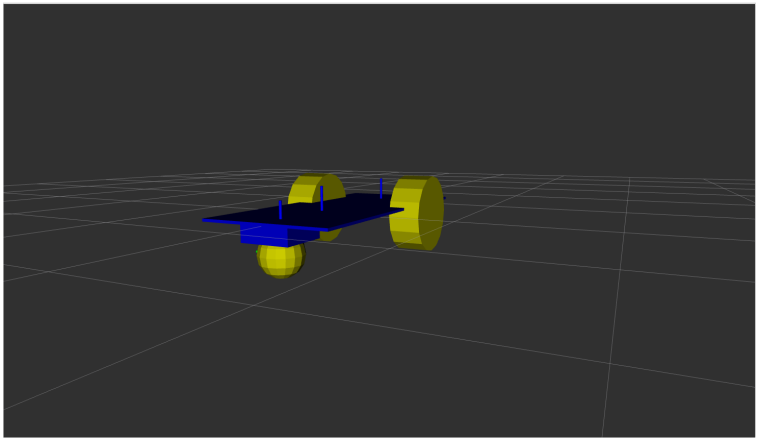

FIGURE 4.3: Second Model

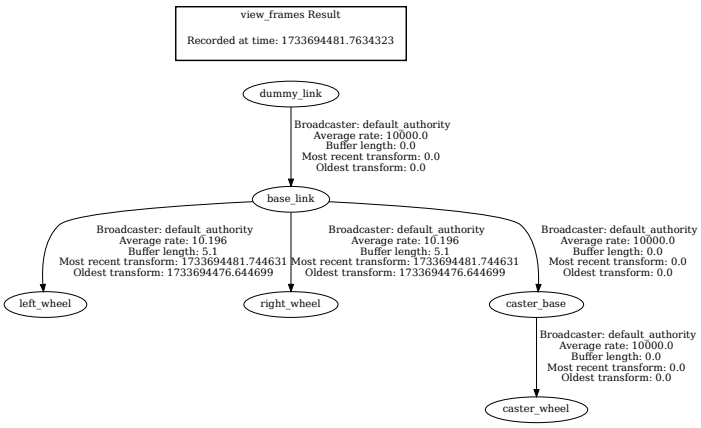

FIGURE 4.4: tf2 tree structure

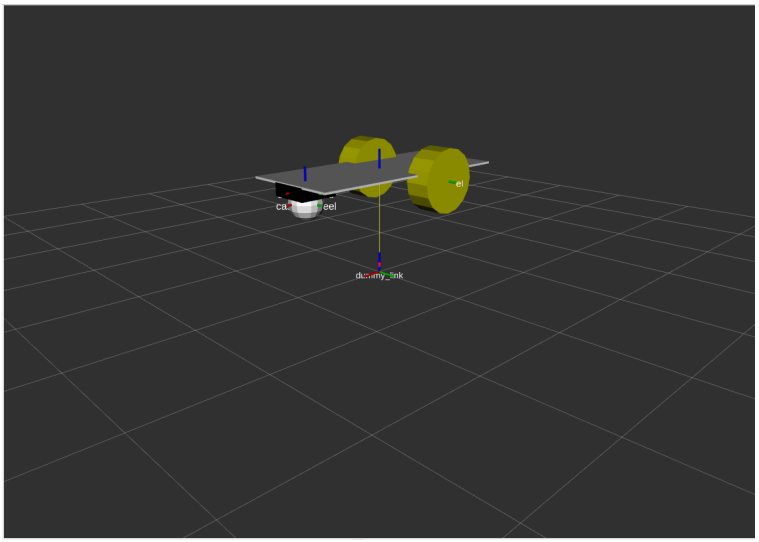

FIGURE 4.5: Final Model

In the final model, the additional link `dummy_link` is added to generate a world-like layer. The idea is to vizualize a world component which could be used as a fix point on which the model could be placed and move around. This idea is then implemented, with in the gazebo simulation, but not further followed for the Rviz2 model. The final tf2 tree structure is shown in the figure 4.4.

4.7 Gazebo

The Gazebo simulator is successfully installed and launched. However, initial attempts to use the simulator fail due to the installation of an outdated version. The simulator is initially installed using the default method, which installs Gazebo Harmonic. Since compatibility between the ROS2 distribution and the Gazebo version is critical, Gazebo Ionic is installed to resolve the issue. The detailed installation prozess can be faund in the installation guide A3. Early attempts to integrate the Gazebo plugin into the URDF file were unsuccessful, as the necessary knowledge to perform this task is not yet acquired at this stage of the project. To address this, the URDF file is converted into an SDF file. This translation allowed the model to be launched successfully in the simulator. The conversion process involved starting an empty world, a default package provided by Gazebo, and placing the URDF model into it. The robot is then successfully positioned on the floor within the simulation. Within the SDF file, the DiffDrive plugin is integrated, enabling the movement of the simulated robot. Forward and backward motion worked seamlessly; however, turning left and right posed challenges.

Also, the attempt to create a connection between the gazebo model and ROS2 failed on multiple occasions, due to the wrong implementation of the `gz_ros_bridge` node. To solve this problem the ROS2 control and ROS2 controller packages could have been used, these packages are widely used in the ros community and offer a variety of control protocols and demonstration models, which show how to implement the packages, in the process of building a control system for the model and real-world robot. Multiple attempts were made to implement the packages and use them to control the model in the gazebo simulation, which all failed. Due to the time restrictions on this project, the attempt to work with this package is postponed to a further project. In the final URDF file, the velocity and dynamics of the model were defined in more detail, and the gz_controll.xacro which contains the differential drive plugin allowed the integration of the gazebo node in the launch file. The final model which is shown in the figure 4.6 could be moved seamlessly in all directions and could be controlled by sending messages to the /cmd_vel topic.

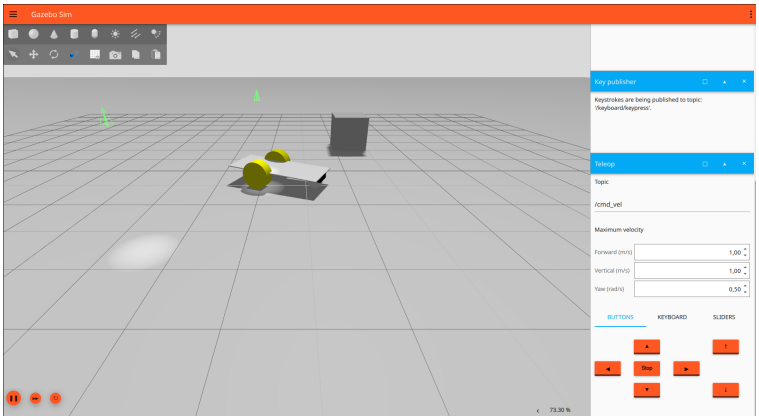

FIGURE 4.6: Gazebo Simulation

4.7.1 Controlling the real robot

The primary remote control system for the realworld robot is developed using two repositories the ros_arduino_bridge and serial_motor_demo. The serial_motor_demo

provided two nodes and accompanying test scripts. The ros_arduino_bridge provided the Arduino scripts. Given that the project's requirements closely aligned with the two repository's existing components, it is possible to utilize the repositories for initial test runs with minimal code modifications. This approach significantly reduced development time within the project. To enable customization without encountering permission conflicts, the repositories were first forked, creating an independent copy. These forked repositories were then cloned onto the control unit, the robot controller. To streamline the process of uploading scripts to the Arduino, the Arduino IDE is installed on the Raspberry Pi 4. A secure SSH connection between the laptop and the Raspberry Pi is established using Visual Studio Code's Remote extension. Additionally, the Arduino extension is installed in the remote environment, facilitating direct script uploads from the Raspberry Pi to the Arduino. The drive node is uploaded from the Raspberry Pi to the Arduino. The initial attempt to control the robot using the provided code is successful, representing a significant milestone in the project's progression.

4.8 Building the digital twin

At this phase of the project, the model could be visualised in Rviz2 and the wheel joints could be moved by using the joint state publisher. The model could be simulated with gazebo and controlled with the differential driver plugin. The real robot's movement could be enhanced by the ROS2 node gui.py and driver.py. The wheels could be moved separately and the speed could be adjusted.

In the first attempt to combine the elements, the ROS2 node `gui.py` is altered. An attempt is made to connect the joint state publisher node with the `gui.py` node. Therefore a message from the gui.py node is sent to the `/joint_state` topic, where the message should be further passed to the joint state publisher node. This resulted in the movement of the wheel joints in the Rviz2 model. However, the model itself remained static. This could be explained by the lack of the dynamic implementation for the Rviz2 model.

A further attempt is made by including the "odom" node. The odometry system provides a locally accurate estimate of a robot's pose and velocity based on its motion. The odometry information can be obtained from various sources among other things the wheel encoders ("Setting Up Odometry — Nav2 1.0.0 Documentation", n.d.). Since the real robot did not include motor encoders a possible solution could have been to use the data representing the encoders of the digital twin.

This attempt is not further investigated, due to the time restrictions and complexity. Since the model staticity could not be overcome within the rivz2 environment the correct implementation of the tool is postponed to the following project. The further development of the digital twin is established in Gazebo. The `gz_ros_bright` node is used to establish a connection between ROS2 and Gazebo. The gui.py node is exchanged with a `self-developed` node. The `cmd_vel_to_motor_command` node translates the speed values from the `/cmd_vel` topic in a format which the motor controller could process. The two messages were sent simultaneously to the model and the real-world robot. The node functions as the interface between the robot and the gazebo simulation. By operating the control panel or sending commands over the terminal the model and the robot could be moved synchronously.

Chapter 5

Discussion

5.1 Achievements

The primary objective of the project, to develop a DT for an autonomous robot, was achieved, albeit with adjustments to the complexity of the digital twin. Throughout the project, a functional URDF file could be developed along with the `cmd_vel_to_motor_command` node. The URDF file could be spawned in the visualization tool rviz2 and the simulator gazebo. A launch file could be developed which started the simulation and visualization tools, along with the necessary nodes to display and control the model. Two motors are the actuators of this project. They are simulated by the differential driver plugin that provides basic movement functionality for the digital twin. Further, the connection between the control unit and the edge device could be established and motor speed commands could be sent and received from the control unit to the edge device. The simulator gazebo and ros2 could communicate over the `gz_ros_bright` enabling the `cmd_vel_to_motor_command` node to send and receive messages from the `cmd_vel_to_motor_command` topics.

5.2 Functionality and Limitations

While the simulation replicated the robot's movement, `real-time` synchronization was not achieved. Furthermore, the digital twin was a simplified representation of the robot, lacking the integration of sensors and environmental elements. The `/cmd_vel` and `motor_command` node facilitated the sending of a command to the motor controller but did not allow the motor controller to transmit feedback messages, such as actual speed or encoder data. In a sophisticated system, the motor controller should be able to send feedback messages. By doing so a node like the odometry node could compare the velocity and speed data of the real-world robot and the digital twin. This comparison is essential for real-time synchronization. However, due to the unsuccessful implementation of the Odom node, the model does not receive feedback messages from the real robot, limiting the ability to perform bidirectional communication and validation. Consequently, the digital twin serves more as a reflection of the robot's state rather than a true twin with bidirectional feedback. The present model does not include any sensors as they increase the complexity of the robot significantly. This complexity is needed to meet the final objective but is beyond the scope of this project.

5.2.1 Challenges and Bottlenecks

The installation phase of this project provided a valuable learning experience as well as one of the main challenges. It involved exploring several methods for installing ROS2 to identify the most suitable approach for the project. This phase was particularly valuable, as it showed the benefits and limitations of different environments and installation techniques. Several environments were tested, including VirtualBox, WSL2, Docker, and Ubuntu 24.04 Linux. While these were the primary tools explored for this step, the testing process allowed for a deeper understanding of how each method operates and interacts with ROS2's ecosystem. This testing phase required significantly more time than initially anticipated, but it deepend the understanding of each environment. Additionally, the understanding of the complex ecosystem of ros2 required substantial effort, making it one of the most time-consuming aspects of the project. One of the challenges regarding the understanding of the ros2 systematics was the installation of additional components. These installations often failed due to version conflicts within the components and packages used. This problem was primarily caused by the choice of a developer distribution instead of a LTS version. The developer distribution is better suited for projects requiring cuttingedge packages or tools not yet part of the LTS version and for developers with substantial experience in the ROS 2 ecosystem. Neither of these conditions was applicable in this project, making the developer distribution an unsuitable choice. The choice of the developer distribution is likely to have contributed to the unsuccessful implementation of ros2_control and ros2_controller. However, this remains uncertain, as other factors, such as limited familiarity with the ros2 ecosystem or configuration complexities, may also have played a role. Further, the time point of the establishment of a connection between the control unit and the edge device was later achieved as anticipated. This caused the project to stagnate.

Chapter 6

Conclusion and outlook

6.1 Conclusion

Overall, the project achieved its fundamental objective but in a simplified form. The digital twin successfully mirrors the robot's behaviour to a degree but lacks real-time synchronization, environmental representation, and sensor integration. Within the project, a primary attempt was made to use the ros2 ecosystem to generate a digital twin. A deeper understanding of the ros2 applications could be established up to an extent, where a functional ros2 node could be developed and the native ros2 communication system could be used to control a real-world robot and a gazebo model. The choice of distribution and installation challenges significantly impacted the project's progress and complexity, but provided a deeper understanding of the configuration within the ros2 ecosystem.

6.2 Outlook

This project served as the cornerstone for a bachelor thesis, which aims to develop an autonomous robot tailored for the envisioned application of maintaining green rooftop ecosystems with photovoltaic panels installed. Key advancements to be addressed based on the project presented, include the integration of feedback mechanisms from the motor controller. This addition would enable data-driven comparisons between the digital model and the physical robot, enhancing precision in control and orientation. Furthermore, the simulation environment should be refined to accurately represent the rooftop green spaces where the robot will operate. For this detailed simulation, an alternative to Gazebo could be a more efficient solution since the gazebo simulator's complexity rises with the complexity of the model. Implementing sensors such as cameras, infrared systems, or lasers would be critical to ensure the robot can effectively perform its tasks. The implementation of neural networks and advanced algorithms would be essential for the development of a robust object detection and decision-making system. To streamline the development process and enhance deployment flexibility, using Docker containers for the ROS2 system would ensure portability, consistency, and simplified setup while enabling efficient cross-device deployment and resource isolation. Finally, the robot design of the presented project would not meet the functional requirements for maintaining rooftop ecosystems. Therefore, a new robot design is be needed that would include a new URDF file for the digital twin.

Bibliography

Albergo, N., Rathi, V., & Ore, J.-P. (2022). Understanding Xacro Misunderstandings. *2022 International Conference on Robotics and Automation (ICRA)*, 6247–6252. https://doi.org/10.1109/ICRA46639.2022.9812349

Dirk, T. (n.d.). *A universal build tool*. Retrieved November 19, 2024, from https://design.ros2.org/articles/build_tool.html

Docker. (n.d.). *Ros - Official Image | Docker Hub*. Retrieved November 18, 2024, from https://hub.docker.com/_/ros

Fahimi, F. (2009). *Autonomous Robots: Modeling, Path Planning, and Control*. Springer US. https://doi.org/10.1007/978-0-387-09538-7

Indri, M. (n.d.). AMR system for autonomous indoor navigation in unknown environments.

Kam, H. R., Lee, S.-H., Park, T., & Kim, C.-H. (2015). RViz: A toolkit for real domain data visualization. *Telecommunication Systems*, *60*(2), 337–345. https://doi.org/10.1007/s11235-015-0034-5

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

Microsoft. (2023, December 5). *Was ist das Windows-Subsystem für Linux?* Retrieved December 11, 2024, from https://learn.microsoft.com/de-de/windows/wsl/about

Newans, J. (n.d.). *Describing robots with URDF | Articulated Robotics*. Articulated Robotics. Retrieved November 19, 2024, from https://articulatedrobotics.xyz/tutorials/ready-for-ros/urdf

Open Robotics. (n.d.-a). *About – Gazebo*. Retrieved November 16, 2024, from https://gazebosim.org/about

Open Robotics. (n.d.-b). *Distributions — ROS 2 Documentation: Rolling documentation*. Retrieved November 13, 2024, from https://docs.ros.org/en/rolling/Releases.html

Open Robotics. (n.d.-c). *Distributions — ROS 2 Documentation: Rolling documentation*. Retrieved November 9, 2024, from https://docs.ros.org/en/rolling/Releases.html

Open Robotics. (n.d.-d). *Jazzy Jalisco (jazzy) — ROS 2 Documentation: Jazzy documentation*. Retrieved November 9, 2024, from https://docs.ros.org/en/jazzy/Releases/Release-Jazzy-Jalisco.html

Open Robotics. (n.d.-e). *ROS: Home*. Retrieved August 14, 2024, from https://www.ros.org/

Open Robotics. (n.d.-f). *Understanding nodes — ROS 2 Documentation: Rolling documentation*. Retrieved December 10, 2024, from https://docs.ros.org/en/rolling/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Nodes/Understanding-ROS2-Nodes.html

Open Robotics. (n.d.-g). *Urdf - ROS Wiki*. Retrieved November 14, 2024, from https://wiki.ros.org/urdf

Open Robotics. (n.d.-h). *Using colcon to build packages — ROS 2 Documentation: Rolling documentation*. Retrieved November 19, 2024, from https://docs.ros.org/en/rolling/Tutorials/Beginner-Client-Libraries/Colcon-Tutorial.html

Rivera, Z. B., De Simone, M. C., & Guida, D. (2019). Unmanned Ground Vehicle Modelling in Gazebo/ROS-Based Environments. *Machines*, *7*(2), 42. https://doi.org/10.3390/machines7020042

Setting Up Odometry — Nav2 1.0.0 documentation. (n.d.). Retrieved December 6, 2024, from https://docs.nav2.org/setup_guides/odom/setup_odom.html

Stączek, P., Pizoń, J., Danilczuk, W., & Gola, A. (2021). A Digital Twin Approach for the Improvement of an Autonomous Mobile Robots (AMR's) Operating Environment—A Case Study. *Sensors*, *21*(23), 7830. https://doi.org/10.3390/s21237830

Takaya, K., Asai, T., Kroumov, V., & Smarandache, F. (2016). Simulation environment for mobile robots testing using ROS and Gazebo. *2016 20th International Conference on System Theory, Control and Computing (ICSTCC)*, 96–101. https://doi.org/10.1109/ICSTCC.2016.7790647

Urdf/XML - ROS Wiki. (n.d.). Retrieved November 19, 2024, from https://wiki.ros.org/urdf/XML

Voas, J. (2021, April 16). *(Draft) Considerations for Digital Twins Standards*. https://doi.org/10.6028/NIST.IR.8356-draft

Wahde, M. (2016). INTRODUCTION TO AUTONOMOUS ROBOTS.

Appendix A

Installation guides

A.1 Change Operating System of Control Unit

A.1.1 Objective

To simplify the work experience, a change in the main operating system on the control unit was performed. The original OS was Windows 10, and the target OS was Ubuntu 24.04 LTS.

A.1.2 Procedure

To perform the reboot of the OS, the following steps were taken:

A.1.3 Preparation: USB Stick and ISO Image

- USB Stick with 128GB.
- Download the Ubuntu ISO: https://ubuntu.com/download.

A.1.4 Step 1: Create a Bootable USB Stick (For Windows Users)

Download and Start Rufus (on Windows)

- Download Rufus: https://rufus.ie/de/.

Configure Rufus Settings

- **Device:** Select the USB stick in the "Device" dropdown menu.
- **Boot Selection:** Click "SELECT" and choose the Ubuntu ISO file downloaded earlier.
- **Partition Scheme:**
  - Select **GPT** when using UEFI.
  - Select **MBR** when using an older BIOS or UEFI with CSM (Compatibility Support Module).
- **Target System:**

  - UEFI (non-CSM) for newer devices.
  - BIOS (or UEFI-CSM) for older devices.
- **File System:** FAT32.

Start the Process

- Wait for the process to complete.

A.1.5 Step 2: Boot Ubuntu from the USB Stick

- Plug in the USB stick and restart your laptop.
- Open the Boot Menu:
  - Use one of the following keys: F12, F10, Esc, or Del.
- Select the USB stick as the boot device to start Ubuntu.

A.1.6 Step 3: Install Ubuntu

Start the Installation

- Follow the installation guide.

Choose Installation Type

- Example: **"Erase disk and install Ubuntu"** to remove Windows and use only Ubuntu.

Advanced Features (Optional)

- Configure as needed.

Confirm Installation

- Wait for the process to complete.

A.1.7 Step 4: After Installation

- Remove the USB stick and restart.
- Use Ubuntu.

A.2 Install ROS2 on Ubuntu 24.04

A.2.1 Objective

The Objective of this experiment was to install ROS2 rolling, on the controll unit (Laptop HP with Ubuntu 24.04) and the Raspberry Pi (also with Ubuntu 24.04). The main issue was to creat a connention between the machines.

A.2.2 Procedure

The first step was to install ROS2 rolling desktop on both machines. For that the instructions from the ros2 website where used. https://docs.ros.org/en/rolling/Installation/Ubuntu-Install-Debs.html

After the installation the source command for ROS2 setup file was added to the .bashrc file on both devices.

```
#oben file via editor
nano ~/.bashrc
# add the command on the botton of the file
source /opt/ros/rolling/setup.bash
```

To creat a connention between the machines the following steps were follwed, on both machines. The ROS_IP and the ROS_DOMAIN_ID where addet after the source command for the ros2 setup file, within the .bashrc file. The ROS IP adresse represented the IP adresse of the current machine, and the ROS DOMAIN ID was the same domain number for both machines.

```
export ROS_DOMAIN_ID=0
export ROS_IP=192.168.x.xx

```

Then the .bashrc file was sourced on both machines with the following command:

```
source ~/.bashrc

```

A.2.3 Testing

To test if the two machines could communicate a demo node of ROS2 was used. Therefore the two commands:

```
ros2 run demo_nodes_cpp talker
ros2 run demo_nodes_py listener
```

Where used, one machine was sending a messeage with the talker, and the other reseved the message with the listener.

A.2.4 Additional

If the connection still can't be astablished the following steps can be tryed:

1. Check if your SSH-Connection works correctly.

2. Check your firewall status

3. When using a virtuell machine, make sure your controll unit is connetet through etahl cable to the network, and not over wlan.

A.3 Install Gazebo Ionic

A.3.1 Objective

The inidial try to install gazebo was by using the defalt installation. Therefore the command:

```
sudo apt install ros-rolling-ros-gz
```

was used. This installed a version von gazebo which was to old for the ubunt 24.04 operation system. Due to this the error

```
[GUI] [Err] [Ogre2RenderEngine.cc:1301]  Unable to create the
rendering window:
OGRE EXCEPTION(3:RenderingAPIException): currentGLContext was
specified with no current GL context
 in GLXWindow::create at ./.obj-x86_64-linux-gnu/gz_ogre_next_vendor-
prefix/src/gz_ogre_next_vendor/RenderSystems/GL3Plus/src/windowing/GLX/
OgreGLXWindow.cpp (line 165)
```

accuret. This leat to the interution of the simulation, before the simulation could be loaded the programm was killed.

A.3.2 Procedure

As describet on the official website of gazebo, a specific version of gazebo had to be installed. For ROS2 Rolling the Gazebo Ionic was compartible. After removing the humble version of gazebo, the ionic version was installed by following the instructions of the official website:

```
https://gazebosim.org/docs/latest/install_ubuntu/
```

Appendix B

Use of AI Tools in the Project

In this project, various AI tools were utilized. The specific tools and their roles are outlined below:

AI Tools

- **DeepL**: Used for translating text
- **Grammarly**: Assisted in correcting grammatical errors and improving the overall readability of the text.
- **ChatGPT**:
  - Assisted in synonym searches
  - Corrected grammar and clarity of written content.
  - Improved the structure and flow of the text
  - Provided explanations for LaTeX functions and structures to streamline the document creation process.
- **ChatGPT and Codeium**: Supported coding tasks by:
  - Improving existing code snippets and structures.
  - Explaining complex C++ code scripts and structures for better understanding and implementation.