Academic Year 2023 – 2024
Place: FEMTO-ST
Department: AS2M
Supervisor(s): Dr. W. Haouas & Dr. S. Dembélé

Complex Systems Engineering

Implementation of ROS-based platform for simulation of quadruped mobile robot

Koulnodji Elysee

July 28, 2024

Abstract

The design, development and validation of robots greatly depends on how satisfactory simulation frameworks are incorporated in research. The project scope covers the use of ROS (Robot Operating System) in simulating a quadruped robot with the help of Gazebo and RViz.

This is achieved by implementing the quadrupedal robot model in Gazebo simulation environment; Introducing controllers for real-time motion control; Integrating camera and lidar sensors for environmental perception, map generation, path planning and autonomous navigation. The sensors output data, the robot dynamics are captured in real time through RViz and this for both the simulated robot and the real robot. This demonstrates the effectiveness of the ROS framework in achieving complex simulation scenarios. The achievements shows advancements in robot simulation using ROS framework and its facilities and could be a bridge to further research in the field of control and robotics.

Keywords: ROS, Gazebo, Rviz, Quadruped Robots, SLAM, Navigation

Contents

List of Figures

Nomenclature

FEMTO-ST	Franche-Comté Electronique Mécanique Thermique et Optique – Sciences et Technologies
AS2M	Automatique et Systèmes Micro-Mécatroniques
GUI	Graphical user interface
ROS	Robot Operating System
ARM	Advanced RISC Machines
CPU	Central Processing Unit
GPU	Graphical Processing Unit
HDMI	High-Definition Multimedia Interface
SLAM	Simultaneous Localization and Mapping
SMART	System.Manufacturing.Academics.Resources.Technologies
LTS	Long Term Support
LIDAR	laser imaging detection and ranging

1 General Introduction

1.1 Description

FEMTO-ST institute is a combined research unit composed of seven departments including the AS2M department. The carried out research in the AS2M department is based on a multidisciplinary foundation combining micro robotics, mechatronics, automation and data science. Among the multiple research interest, the department is engaged in a research initiative centered around the field of land mobile robotics, specifically tailored for local delivery applications across civil and industrial domains.

This project was developed at the SMART Lab within the AS2M department under the supervision of Dr. W. Haouas and Dr. S. Dembélé.

1.2 Problem Statement

In recent years, the field of robotics has shown high advancement due to the revolution of advanced algorithms, improved hardware technologies, and powerful simulation tools Xu et al. [2021].

Simulation of Robotics system design is a crucial phase to ensure efficiency and precision during the development. Multiple simulation frameworks have been used and research has been carried out for the purpose. Recently, Pastor et al. [2021] researched and compared two methods,genetic algorithm and topology optimization, to optimize a quadruped robot through V-rep software by Coppelia Robotics. A year back Chang et al. [2020] researched on gait Planning of Quadruped Robot Based on ROS and the work was simulated and verified by Matlab tool.

Though the utilization of these frameworks has advanced the field, complexity issue was encountered as well as flexibility. However, Robot Operating System (ROS) stands out as a flexible and frequently used platform for robotics research and development.

Quadruped robots are well known for their ability to overcome the multiple dynamics for navigating in complex environments, and performing risky tasks such as rescues, hazardous environment exploration,and military operations Biswal and Mohanty [2021], Raibert et al. [2008]. However, because of their complex dynamics and motion designs, quadruped robots are difficult to develop, control, and test.

Thus, to address these challenges, simulation promoted as suitable solution for prior testing of algorithms, and performance before deployment Matta-Gómez et al. [2014].

The AS2M department offers facilities among which there is an robot, the Unitree A1 quadruped robot, a dog-like robot Fig.2 that runs on a real time Linux(Ubuntu) operating system and is equipped with various components, including an ARM CPU, NVIDIA GPU, stereo depth camera, and optional support for other sensor. The architecture of the control board is showing Fig.1.

Information regarding the architecture of the development board can be found in the software support file Sar [2020] which contains the IP address, Wired and Wireless network and their connection with each other to speed up development. The preliminaries of the robot manipulation are also listed in the quick start guide Sof [2020].

The robot supports ROS Melodic Morenia, the twelfth ROS distribution release for Ubuntu 18.04.

The aim of this project is to implement a ROS platform to control and simulate the existing quadruped robot from Unitree robotics in different scenarios for both real experiment and simulation environment such as Gazebo for realistic interaction of the robot with its surroundings, and RViz a visualization tool for monitoring and evaluating the robot's behavior in real time.

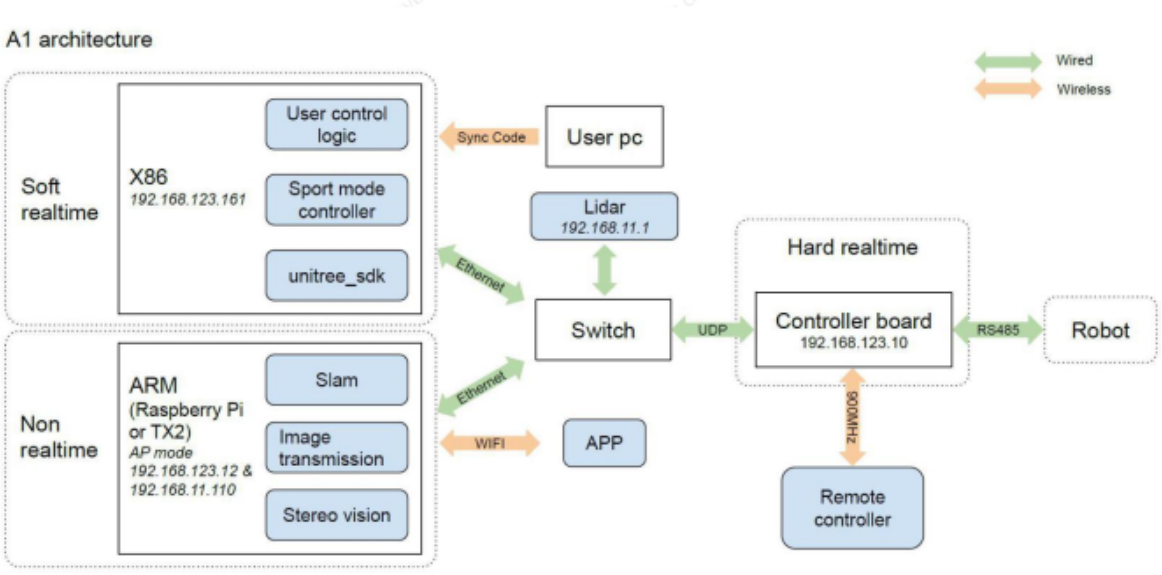


Figure 1: Robot system schematic

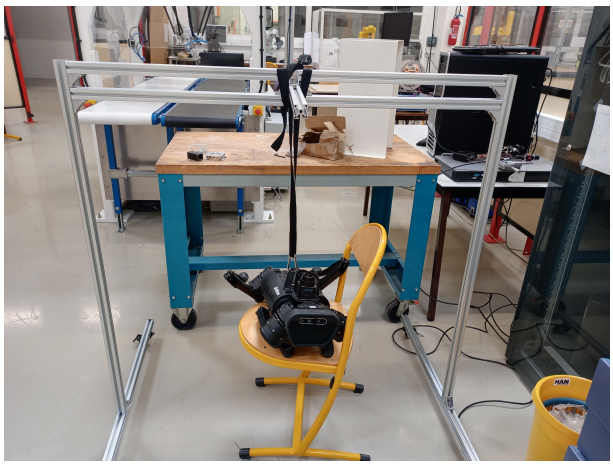


Figure 2: A1 unitree quadruped

1.3 Objectives

The objectives of this project are as follows:

- To install the appropriate operating system, ROS distribution and its dependencies.
- To familiarize with the software frameworks
- To implement the appropriate robot description
- To implement robot controller
- To implement sensors for its feedback measurement
- To control the robot in the simulation environment
- To establish wireless communication
- To implement SLAM and Navigation

1.4 Outline

The remaining part of the project will fall under the following guidelines. First there will be a section that covers the methodology tackling the material and method used to accomplish the objectives highlighted previously, further there will be a section that presents the results obtained out of the applied methodology and a discussion of this results, and then there will follow a section that concludes the overall funding and suggests the future work to be investigated. Finally, an appendix for further information, and annexes with a step-by-step process for the simulation and real experiment.

2 Methodology

2.1 overview

This section presents the material and method for software implementation, robot description implementation and control.

2.2 A1 quadruped robot architecture

The A1 consists mainly of control systems, communication systems, and power systems.
Developing reserved power interfaces consists of power output(5V,12V,19V), and power input 24V. Developing reserved communication is composed of two TX2 USB ports, two Ethernet interfaces, two MiniPC USB ports, on TX2 HDMI and one MiniPC HDMI.
The robot has two built-in hosts: Onboard built-in host Raspberry Pi 4 for real-time communication and an optional built-in host NVIDIA TX2 that can be used for visual SLAM and gesture recognition.
In addition a depth camera allows the quadruped robot to perform real-time image transmission and an optional lidar Fig.3, that allows dynamic obstacle avoidance, navigation planning, autonomous positioning, and map construction.

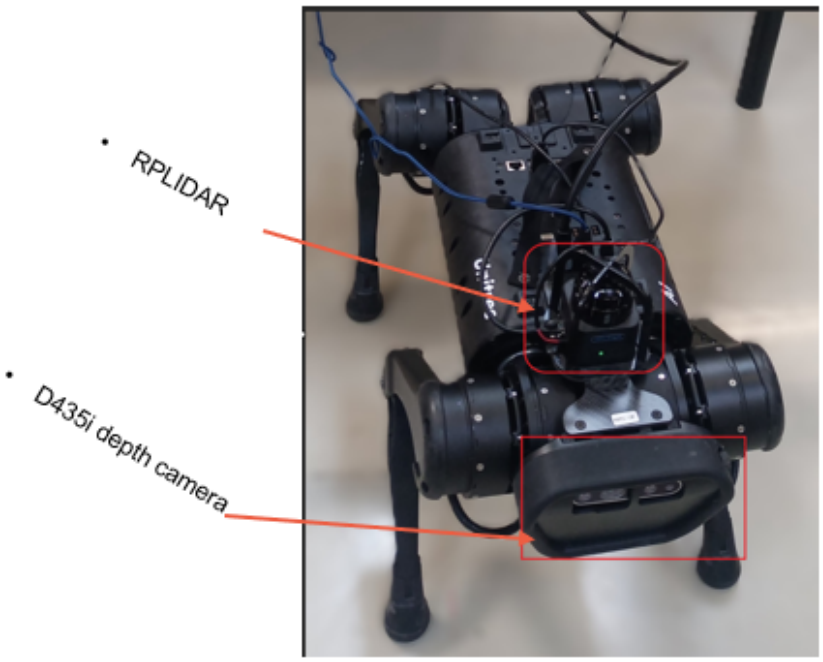


Figure 3: Camera and Rpliar

2.3 Operating system implementation

An open-source software Linux operating system **Ubuntu 18.04 LTS** was used. The selection of Ubuntu 18.04 LTS is based on its compatibility with the robot own by the department, as well as its suitability for long-term projects.

2.4 ROS implementation

For this project, **ROS Melodic** is used because of its compatibility with Ubuntu 18.04 LTS. The installation of ROS Melodic is based on the instructions provided by the ROS community, which include configuring

the relevant environment variables. a pipeline of the installation is shown in Fig.4

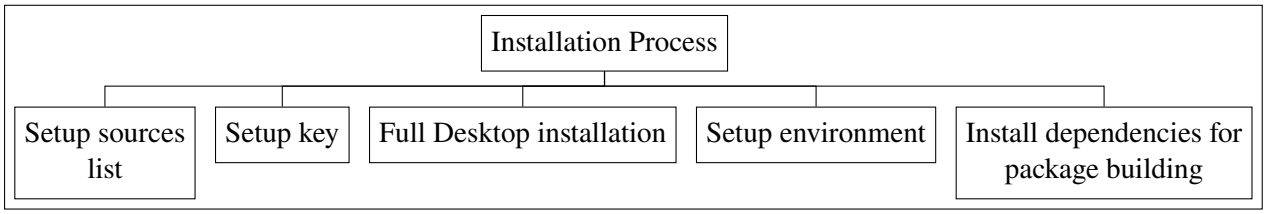


Figure 4: ROS configuration and Desktop-Full Installation

2.5 Network Configuration

The robot control board can be access through a cryptographic network protocol SSH from a Host computer by the help of a wireless communication as highlighted in Fig.5.

It is essential to establish a wireless rather than wired connection from the host machine to the robot so that the robot can perform walking while being controlled from the host PC.

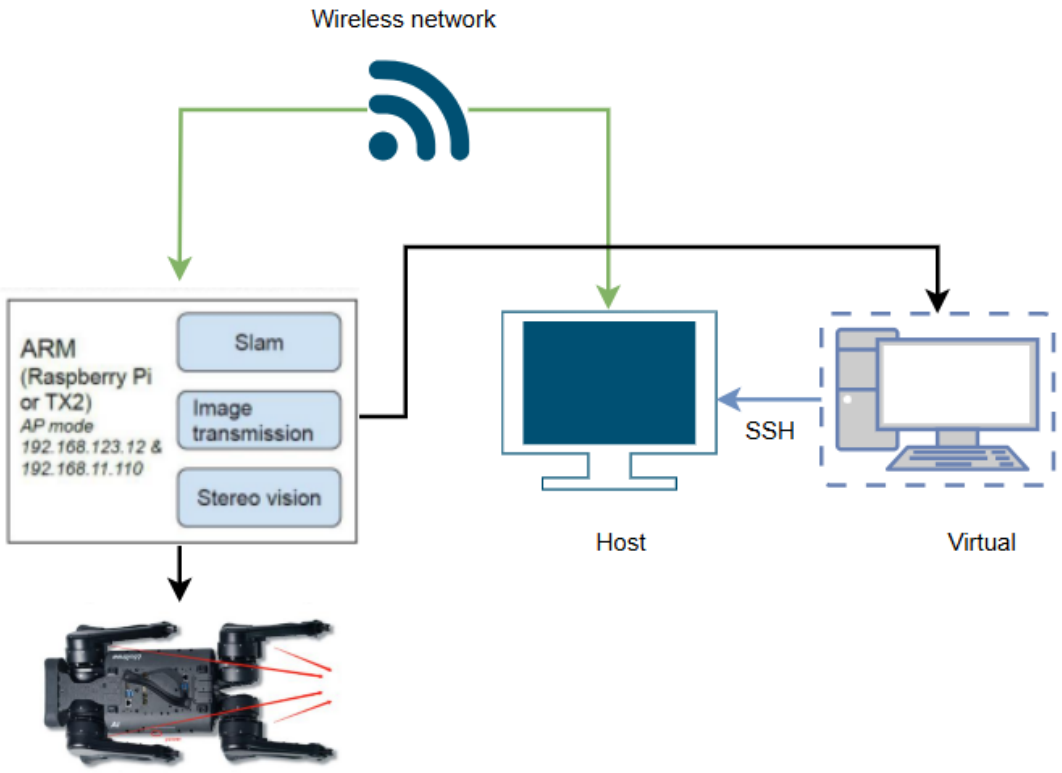


Figure 5: Camera and Rplidar

Detailed procedure can be found in Annex B

2.6 Virtual Robot

2.6.1 Robot description implementation

This project uses a robot model from the "unitree_ros" GitHub repository of the Unitree Robotics company UnitreeRobotics. The repository provides packages for interfacing with the virtual robot models Fig.6, including the one owned by the department, making it a suitable choice for this project.
This speeds up the overall development experience by enabling a smooth and supported primary processes integration.

Figure 6: GitHub repository of the Unitree Robotics company

2.6.2 Controller Implementation

In order to control the quadruped robot within the simulation environment, a teleoperation (teleop) controller is implemented using the **teleop_twist_keyboard** package. Further more a custom controller adapted to the dynamics of the robot is developed on the basis of the controllers provided in the "controllers" directory of the Unitree_simulation" GitHub repository".
The controller was developed following the method of Sen et al. [2017]. These costume controllers are integrated into the ROS framework to enable real-time control of the simulated robot's motion 19. The process is as follows:

Configuration

- Move files holding Robotcontroller lib , inversekinamtics, robotutilities and the python file for the controller to the src folder under the unitree_controller package.

- Adapt file permissions and edit Python scripts to ensure compatibility with Python 2.

ROS Package Configuration

- Update the 'robot_controller.launch' file under 'launch' folder.

- Edit the 'CMakeLists.txt' files to include 'rospy' and 'sensor_msgs' as 'catkin REQUIRED COMPONENTS' and update the 'Package.xml' accordingly to confirm that the appropriate dependencies are provided.

Configuration File and Xacro Compatibility Updates:

- Move the controllers.yaml to the config folder of robot description and update the file for compatibility purpose with the normal.launch file.
- Similarly update the xacro files to be compatible.
  Here below in Fig. 19 a representation of tree structure of the unitree_controller directory.

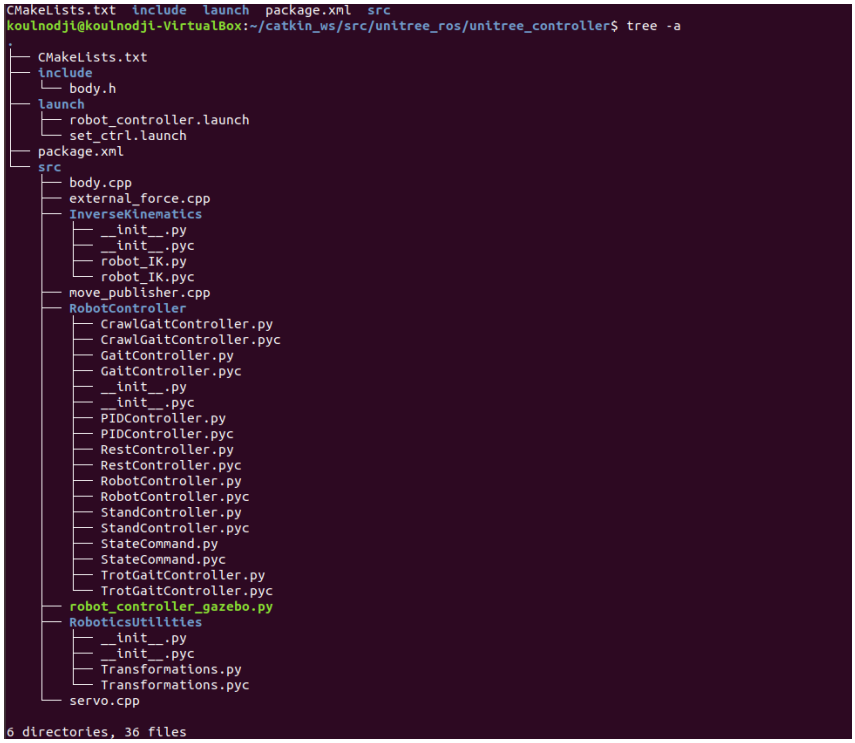


Figure 7: Tree structure of Unitree_controller directory

Detailed procedure can be found in Annex A

2.6.3 Sensors implementation

The ROS interface of the unitree robot did not include external sensors, therefore in order to add sensors to the robot, It is crucial to refer to the Unitree Robotics documentation which gives information regarding the requirements and setting up for the different sensors to be implemented. The steps are:

- Insert the gazebo_plugin of each sensors
- Insert the sensors meshes files
- Insert the sensors xacro macro files
- Modify the Robot xacro file to include the external sensors.

The tree structure of the robot description directory is shown in Fig 8. This structure gave details about the directories and subdirectories holding the necessary files to accomplish the task.

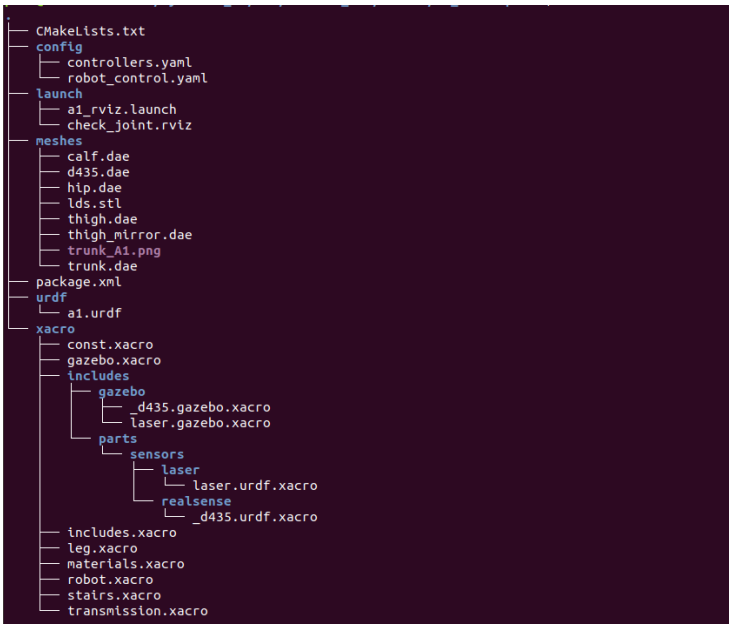


Figure 8: Tree structure of robot description directory

Launch File Update

The main launch file is updated to include the additional nodes:

```
    <!-- load NEW unitree_controller -->
    <include file="$(find unitree_controller)/launch/robot_controller
        .launch"/>

    <!-- New lines added for TWIST MUX-->
    <include file="$(find quadruped_unitree)/launch/custom_twist_mux.
        launch"/>
```

The first line is a comment line to make the program readable, the second line helps to include under a directory called unitree_controller a launch file named robot_controller inside the launch subdirectory. Similarly to the previous, the fifth line includes the launch file named **costum_twist_mux**.

Launch commands

To perform the launch command under the terminal:

- Spawn the robot model on gazebo simulator and Rviz

```
roslaunch unitree_gazebo normal.launch rname:=a1 wname:=earth
```

Where the ‘rname‘ means robot name, The ‘wname‘ means world name, which can be ‘earth‘, ‘space‘ or ‘stairs‘,’no_roof_small_warehouse’.

- unitree_controller

```
rosrun teleop_twist_keyboard teleop_twist_keyboard.py cmd_vel:=/key_vel
```

Detailed procedure can be found in Annex A

2.6.4 SLAM and Navigation

In order to perform simultaneous location and mapping, a ROS package **hector_slam** was used, it contains hector_mapping which is SLAM node, hector_geotiff that help saving the map and robot trajectory to geotiff images files and finally hector_trajectory_server for saving of **tf** based trajectories.

To estimate the position and orientation of a robot withtin a generated map, the Adaptive Monte Carlo localization (AMCL) algorithm combines the laser scan data with the map, detecting any drift that may have occurred during dead reckoning. It then publishes the transformation data which, once taken into account, gives an accurate estimate of the robot's position.

The navigation motion is generated by move_base, who maintains a global and a local cost maps so it can create global and local plans. Its behavior is defined on a set of yaml files namely:

```
base_local_planner_params.yaml
costmap_common_params.yaml
dwa_local_planner_params.yaml
global_costmap_params.yaml
local_costmap_params.yaml
move_base_params.yaml
```

In this project 4 launch files were created, first for starting gazebo and rviz node with robot description, secondly a node to start slam, thirdly a node to start the amcl node and finally a launch file to start the move_base node.

Detailed procedure can be found in Annex A

2.7 Real Robot

2.7.1 Communication set up

In order to enable the pushing and subscribing to topics among different nodes, a configure was done.

The robot in this case acts as the master and the host PC as slave, to do so the **.bashrc** file was edited:

```
    ROS_MASTER_URI=http://192.168.132.12:11311
    ROS_IP=192.168.132.12
```

And the host PC as slave:

```
    ROS_MASTER_URI=http://192.168.132.12:11311
    ROS_IP=0.0.0.0
```

Thanks to this configuration, all the publish data from the robot nodes are retrieved through the host PC for real time visualization.

2.7.2 Setting Up the RPLIDAR

To utilize the lidar, an Ethernet cable with RJ45 connector was used for data transmission and a USB cable for powering the lidar Fig.9. To perform slam, 2 terminals and launching consecutively:

```
 roscore
roslaunch slam_planner slam_planner_online.launch
```

Detailed procedure can be found in Annex B

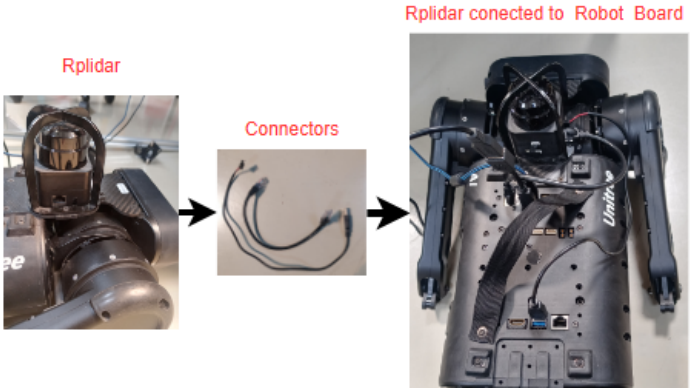


Figure 9: Rplidar connected to robot development board

2.7.3 Setting Up the Camera

The Intel RealSense ROS was used, installed with the command:

```
sudo apt-get install ros-$ROS_DISTRO-realsense2-camera
```

Next, transform the static frame to be connected to each other using **tf** package such as:

```
<node name="base2camera" pkg="tf" type="static_transform_publisher"
      args="0 0 0 0 0 1 /base_link /camera_link 100" />
```

And shown is Fig.10 Then to visualize the camera data run:

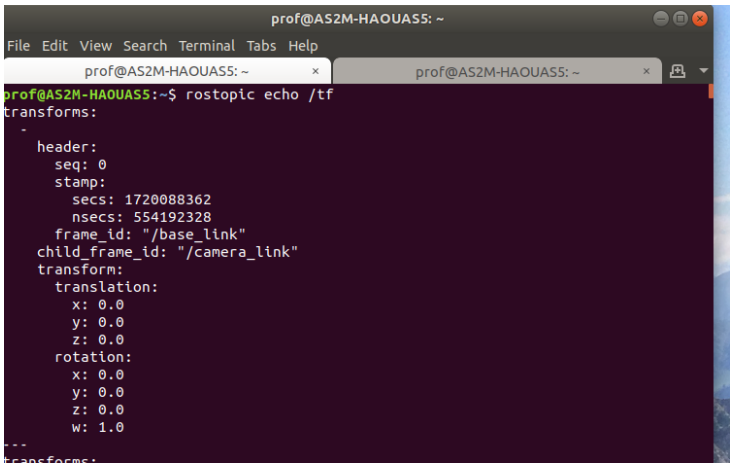


Figure 10: camera to base_link transformation

```
roscore
roslaunch slam_planner slam_planner_online.launch
roslaunch realsense2_camera rs_camera.launch
```

Detailed procedure can be found in Annex B

3 Results and Discussion

3.1 Overview

This highlights the results of the project and a discussion of the findings.

3.2 Simulation Results

3.2.1 Original Robot model

The provided robot model under unitree_ros repository was successfully deployed on gazebo simulator and the visualization done in rviz. The original robot performs only low-level simulation such as stand, rotation around fixed coordinate as highlights in Fig.11.

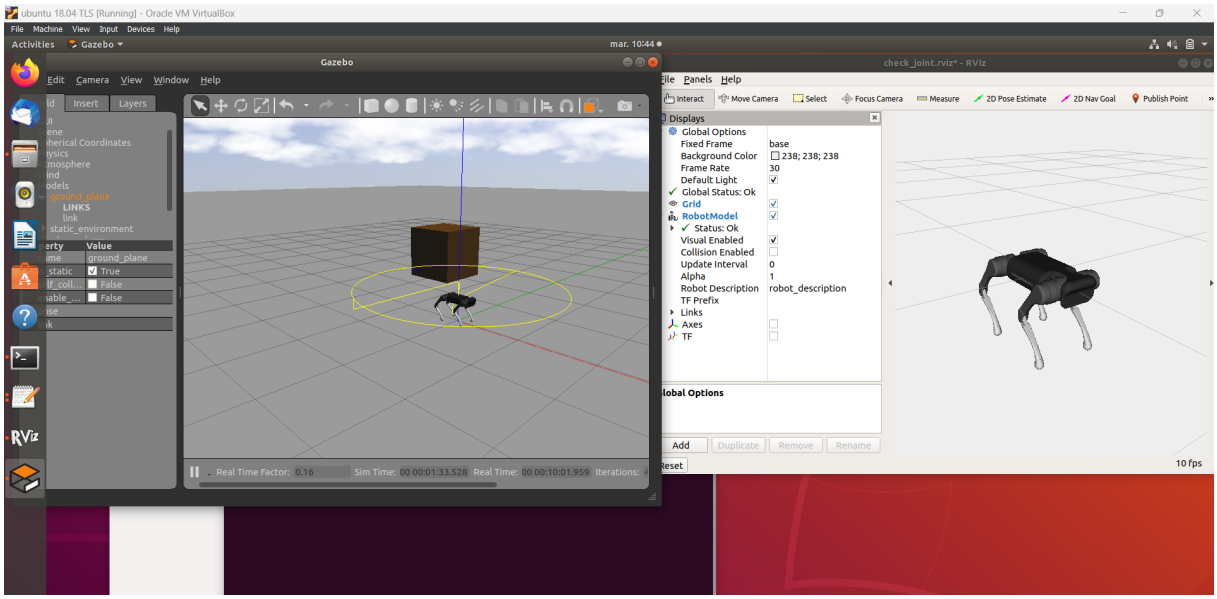


Figure 11: Original robot model on gazebo simulator

3.2.2 High-level simulation

The controller implemented enabled to successfully perform the walking task under the simulation environment gazebo and visualize the output in real-time with the help of rviz tool. The control was done thanks to Teleoperation using keyboard facility. The teleop_twist_keyboard package enabled to send twist message to the robot to control the motion. The Keyboard shortcuts for commanding the speed and tuning values are also displayed Fig.12.

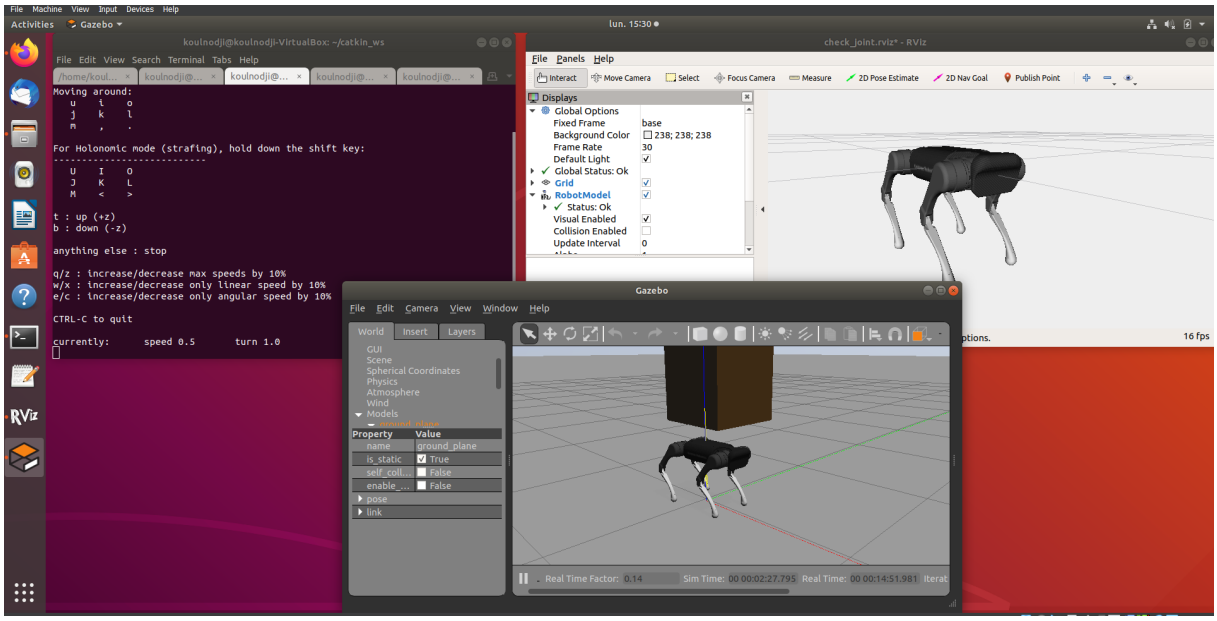


Figure 12: High-level simulation with gazebo and rviz

3.2.3 Realsense camera

The Intel Realsense d435 camera was attached to the robot for visualizing the environment which is crucial for further control algorithm development. The camera was attached to the front trunk of the robot and successful display the environment as shown in Fig. 13

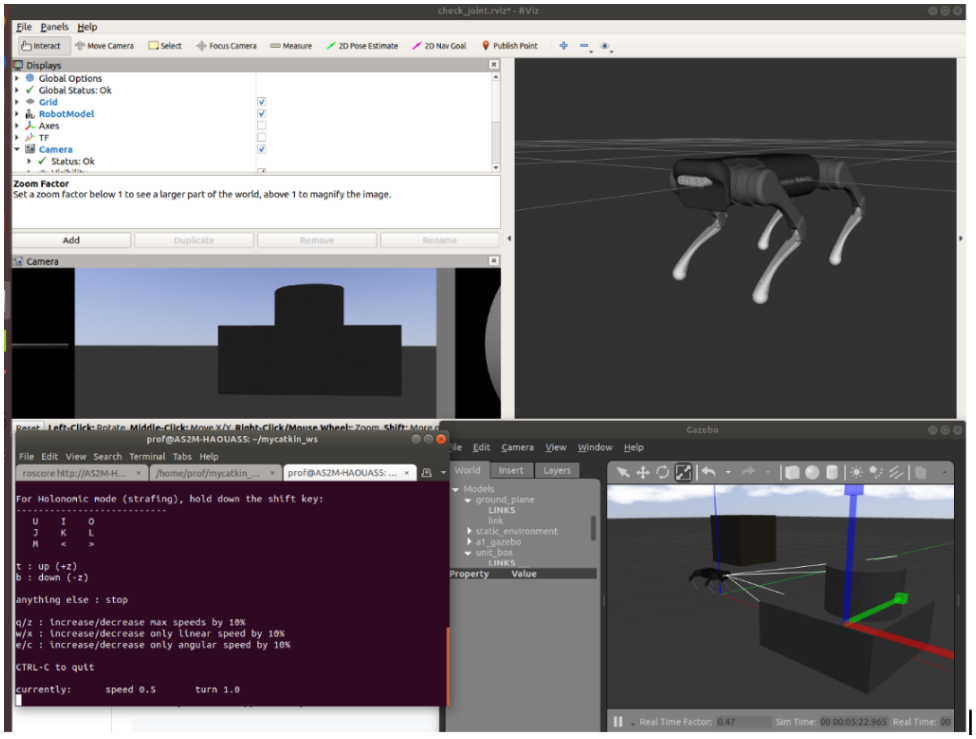


Figure 13: Intel Realsense d435 camera for environment perception

3.2.4 LIDAR sensor

LiDAR sensor integrated scans the surrounding environment and could provide distance measurements to objects which could be crucial for safe navigation in complex environments. The LiDAR sensor attached is shown in Fig.14

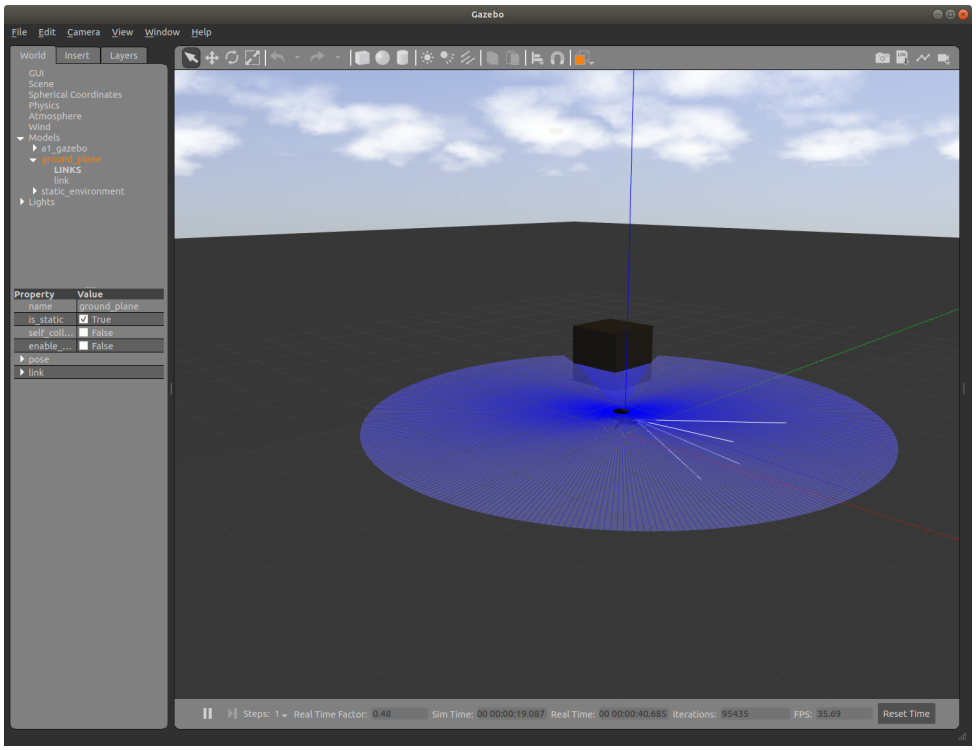


Figure 14: LiDAR sensor scanning environment

3.2.5 Complete simulation with sensors attached

The Full robot simulation under gazebo, visualization under rviz tool and control interface are shown in Fig.15
The lower-right part shows the robot under the simulation environment on it were attached the LiDAR for scanning the surrounding and measure the distance to objects.
The Lower-left part displays the teleop_twist_keyboard that sends the twist message to the robot nodes to perform motion.
The upper-right part is the visualization of the robot through rviz and finally the upper-left displays the output of the camera capturing the robot environment.
The image is displayed in real-time. This is important for further motion planning and autonomous control.

3.2.6 Map generation

The environment captured in 2D map using Laser Scan topic Fig.16

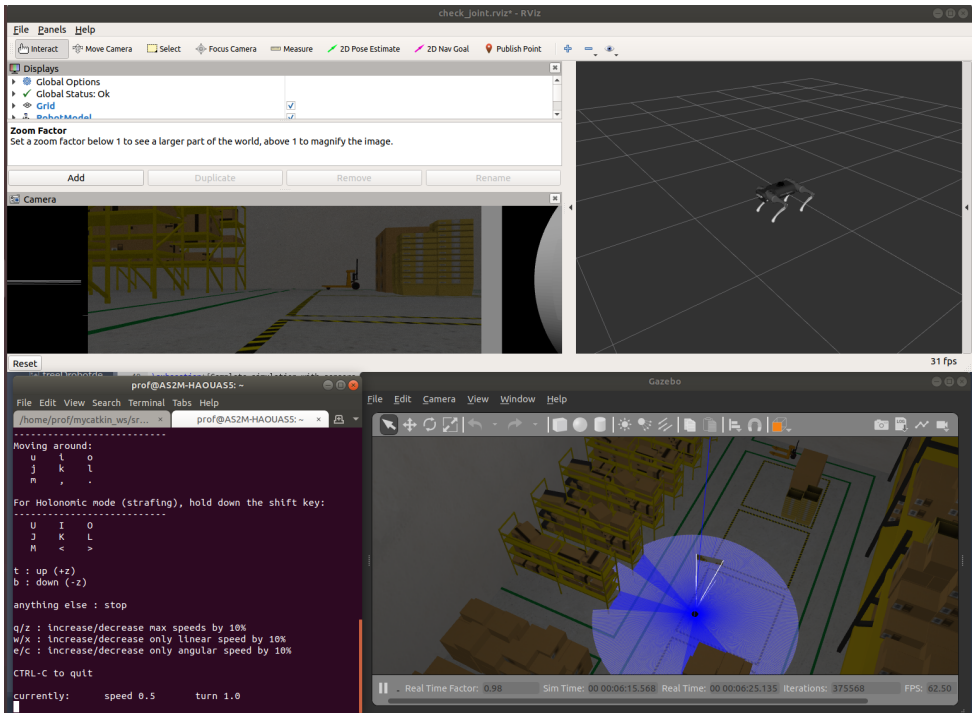


Figure 15: Full simulation with sensors integrated

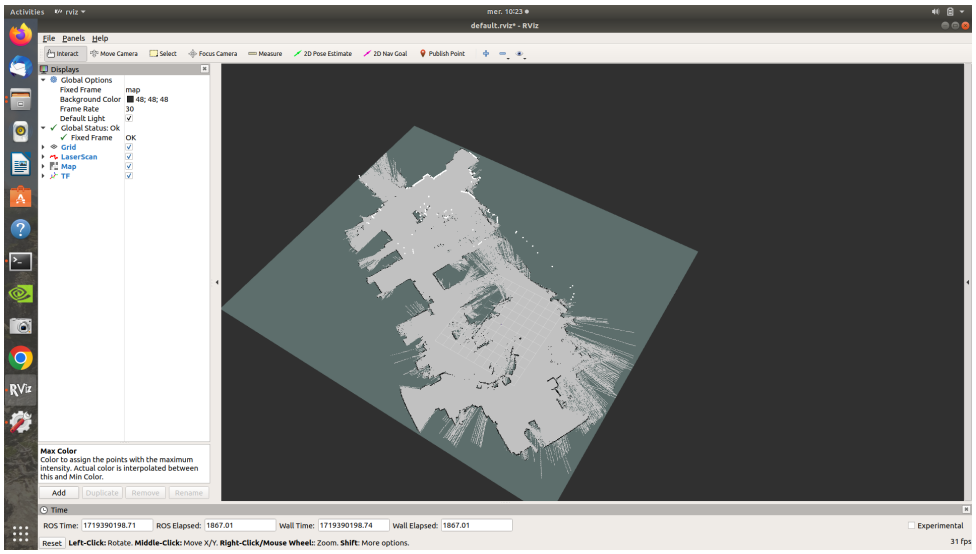


Figure 16: Generated map of SMART Lab

The rqt_graph, ROS GUI tool enables to visualize the graph of interconnections.
The ROS nodes communicate using default transport method used in ROS called TCPROS.
More details can be seen in the **Appendix A, B, and C** where the topics are mentioned in a rectangle and the nodes are represented in ellipses.

3.2.7 navigation and mapping of simulated robot

The Fig.17 shows the path of the simulated robot which navigates autonomously from a world starting point to a desired position while avoiding obstacles.

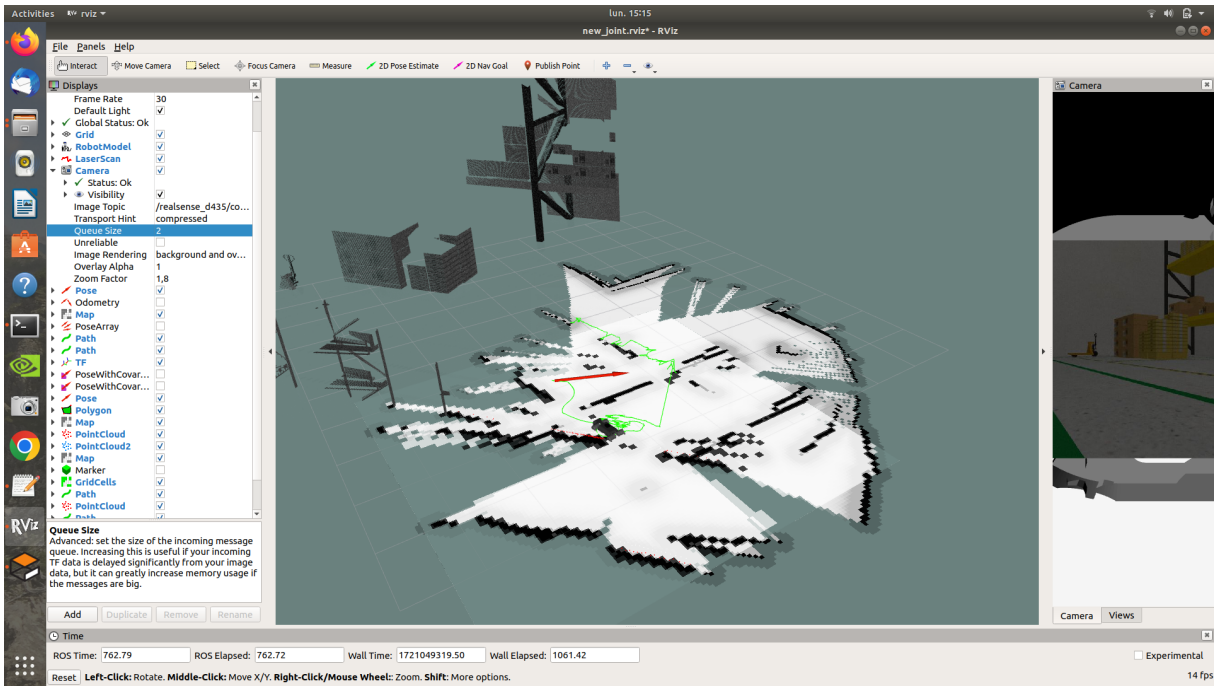


Figure 17: Navigation and mapping of simulated robot on rviz

3.3 Real experiment Results

3.3.1 Map generation and Camera perception

Generated map and environment visualization can be observed in Fig.18.

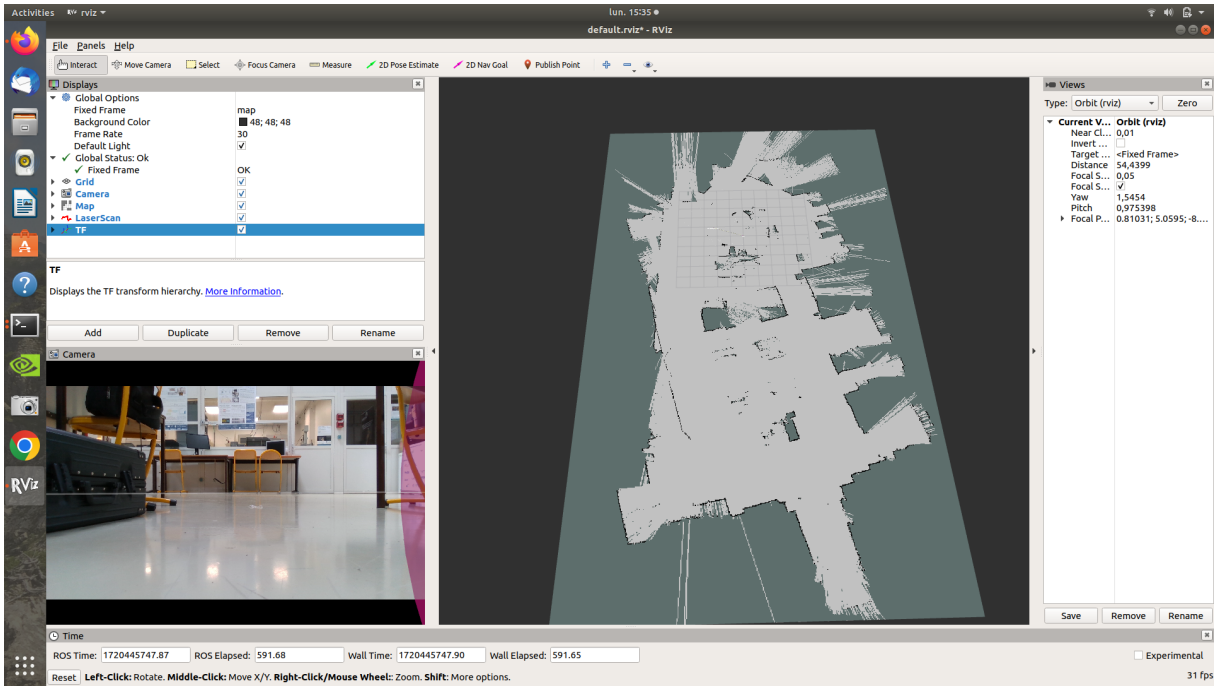


Figure 18: SLAM on real robot with 2D camera

3.4 Discussion

The project funding resulting in successful implementation of the simulation model could be seen as a step further in the development of simulation framework for quadruped robots using the ROS framework.

The combination of ROS, Gazebo, RViz is one of the innovative simulation technologies that revolted the robotic field . These are of great importance for researchers as they offer an environment which is flexible for design, test as well as validate systems on robotics virtually. The success of deploying the quadruped robot model, implementation of controller and integration of sensors demonstrated the capabilities of the ROS framework on robots simulation.

The integration of Intel RealSense cameras as well as LiDAR sensors offered additional capabilities within simulation environment. This gives a wide possibility to program tasks such as obstacle detection, environmental monitoring by the information accumulated on surrounding.

4 Conclusion

The project aiming to help familiarizing with the robot operating system, understanding the core concepts and comprehending the hardware and software requirements for a successful simulation and control through gazebo and rviz.

On the light of the fixed objectives, the methodology used in this project has led to significant advances in the simulation of the quadruped robot using ROS framework. The successful deployment of the robot model, the implementation of controllers and the integration of the Intel RealSense camera and LiDAR sensors have demonstrated in one hand that the methodology used was a success and in the other hand the utility of simulation framework to improve perception capacities, control, and interpretation. In addition the autonomous navigation with obstacles avoidance was investigated in this project and the results demonstrate the capabilities of ROS in developing robotics systems. Overall the project enabled to gain a significant knowledge on robotics system, industrial network, simulation framework tools such as ROS, gazebo and rviz which are among the emerging solutions in robotics field. These results provide a basis for further research and development in the field of robotics simulation, with potential applications in education, research and the robotics industry.

Beyond the outcome of this project, few limitations could be addressed for further improvement.

- Analysis of simulation sensor data.
- Updating the computational resources for complex simulation scenarios.
- Improving navigation algorithms for further optimization of path generation and obstacles avoidance.

References

A1 quick start guide 7.29. Technical report, 2020.

Unitree a1 software guide v2.0. Technical report, 2020.

Priyaranjan Biswal and Prases K Mohanty. Development of quadruped walking robots: A review. *Ain Shams Engineering Journal*, 12(2):2017–2031, 2021.

Shuai Chang, Haibo Du, Yongzheng Cong, Feng Xie, and Jinfeng Zhang. Gait planning of quadruped robot based on ros. In *2020 7th International Conference on Information, Cybernetics, and Computational Social Systems (ICCSS)*, pages 761–766. IEEE, 2020.

Antonio Matta-Gómez, Jaime Del Cerro, and Antonio Barrientos. Multi-robot data mapping simulation by using microsoft robotics developer studio. *Simulation Modelling Practice and Theory*, 49:305–319, 2014.

Robert Pastor, ZDENKO BOBOVSKY, PETR OSCADAL, JAKUB MESICEK, MAREK PAGAC, Erik Prada, LUBICA MIKOVA, and Ján Babjak. Optimizing a quadruped robot: a comparison of two methods. *MM Science Journal*, 2, 2021.

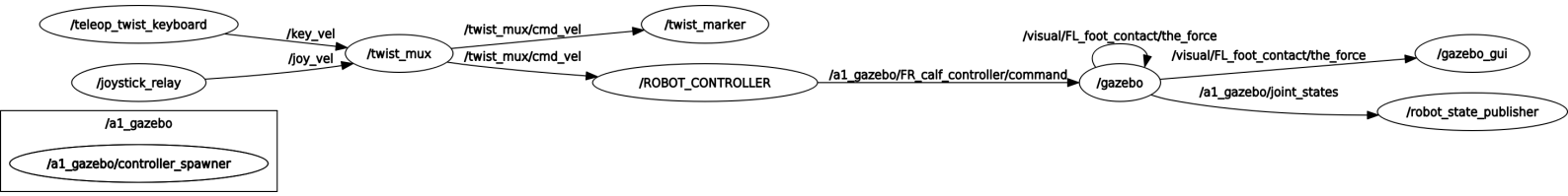
Marc Raibert, Kevin Blankespoor, Gabriel Nelson, and Rob Playter. Bigdog, the rough-terrain quadruped robot. *IFAC Proceedings Volumes*, 41(2):10822–10825, 2008.

Muhammed Arif Sen, Veli Bakircioglu, and Mete Kalyoncu. Inverse kinematic analysis of a quadruped robot. *International journal of scientific & technology research*, 6(9):285–289, 2017.

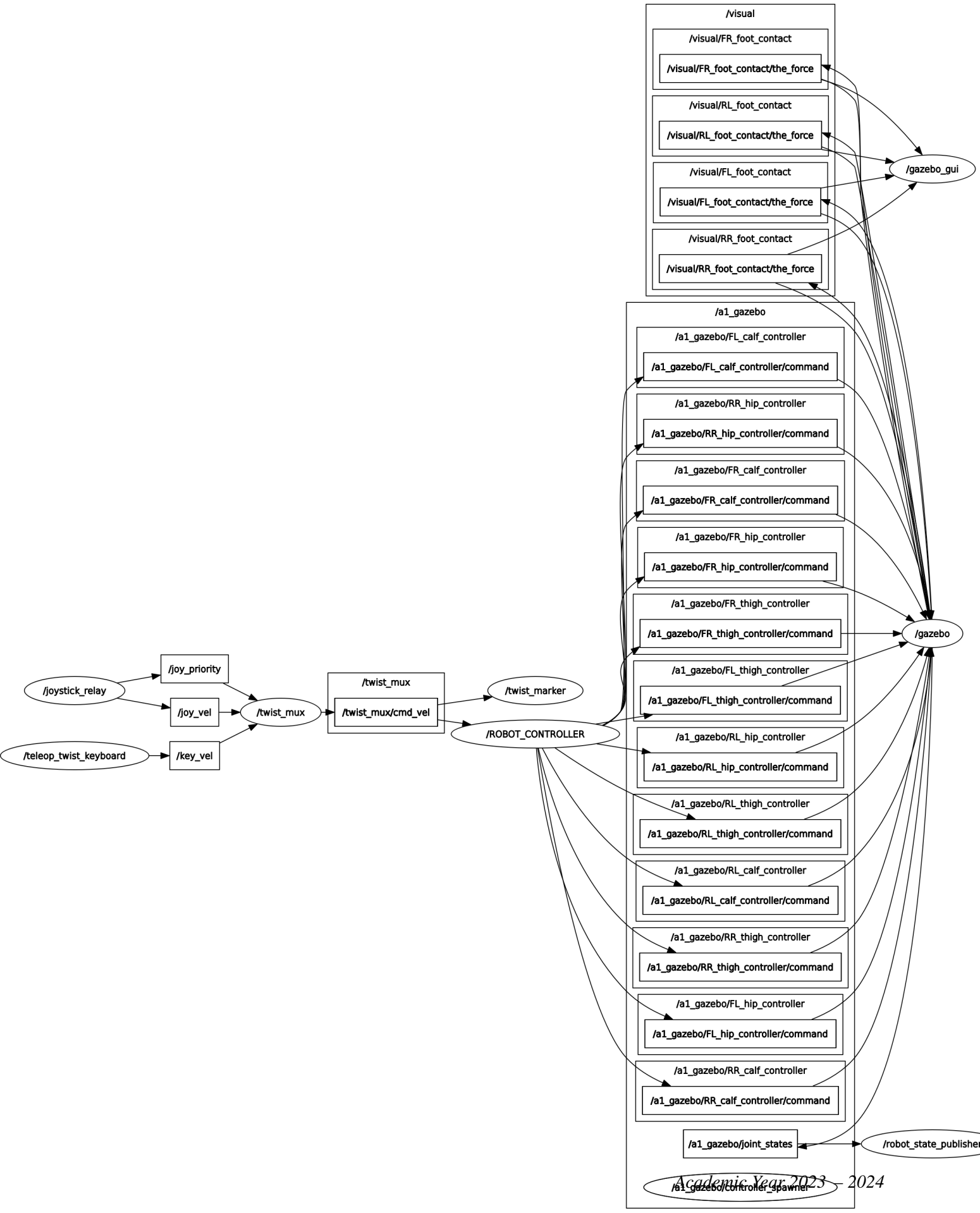
Yongjun Xu, Xin Liu, Xin Cao, Changping Huang, Enke Liu, Sen Qian, Xingchen Liu, Yanjun Wu, Fengliang Dong, Cheng-Wei Qiu, et al. Artificial intelligence: A powerful paradigm for scientific research. *The Innovation*, 2(4), 2021.

5 Appendix

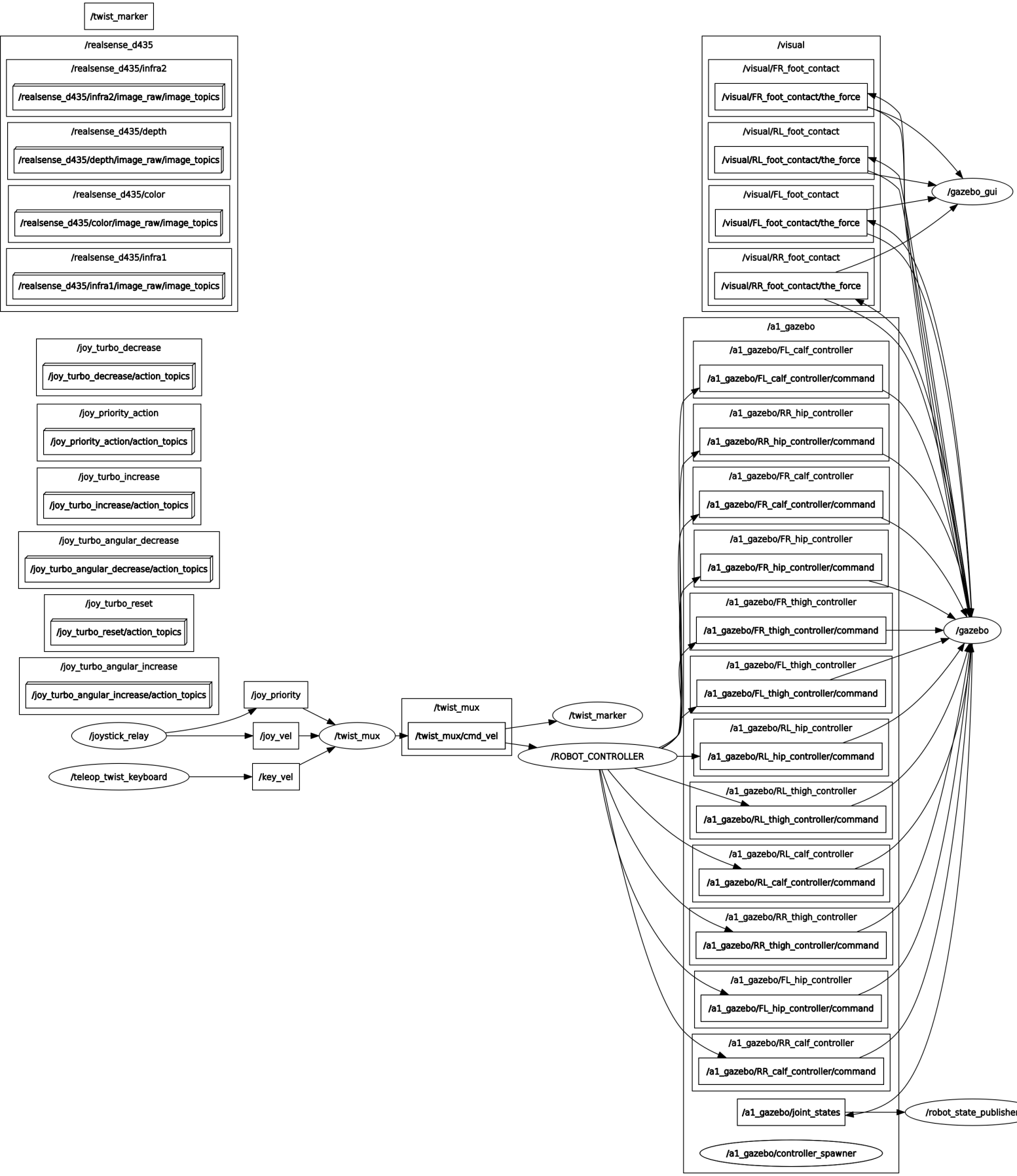
Appendix A :Nodes graph



Appendix B :All Nodes/Topics graph



Appendix C :All Active Nodes/Topics graph



6 Annexes A: Simulated Robot

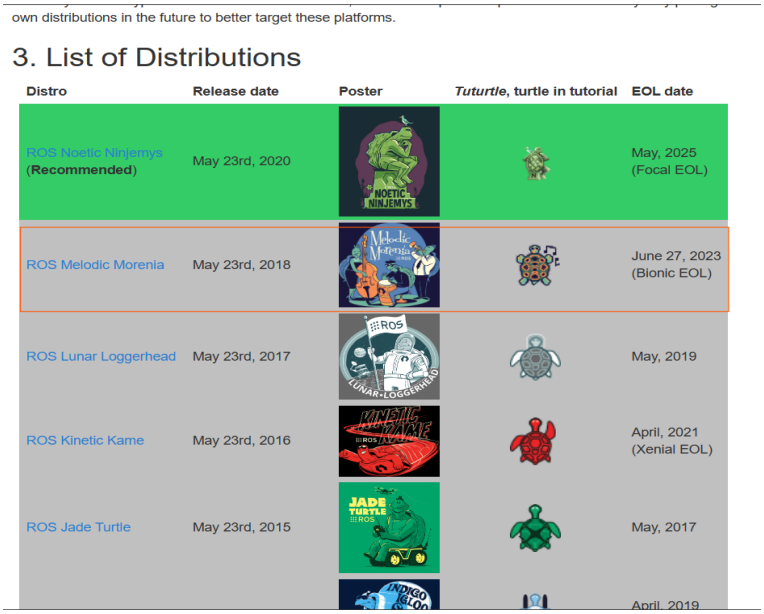
6.1 Overview

This section details the steps and guidelines for installing the software, debugging. A step by step guide for repositories cloning and development, building and update packages.
It will take you through everything required to succeed to perform high-level simulation of the quadruped robot (A1 Uniree) and how to attach camera and LiDAR

6.2 ROS and Dependencies installation

We assume that Ubuntu distro installed is pre-installed. We will go directly to the ROS installation:

Open a new terminal, and from `https://www.ros.org/`, click on ROS Wiki/distribution and select ROS Melodic Morenia here: `https://wiki.ros.org/Distributions`



You can now follow the installation procedure for Ubuntu. Few step to be done, under you opened terminal run this commands:
Setup your computer to accept software from `packages.ros.org`.

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu
$(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'
```

Set up your keys

```
    sudo apt install curl # if you haven't already installed curl
curl -s https://raw.githubusercontent.com/ros/rosdistro/master/ros.asc |
sudo apt-key add -
```

First,update the debian package index with: `sudo apt update`, then choose Desktop-Full Install: (Recommended) : ROS, rqt, rviz, robot-generic libraries, 2D/3D simulators and 2D/3D perception.

```
    sudo apt install ros-melodic-desktop-full
```

It's convenient if the ROS environment variables are automatically added to your bash session every time a new shell is launched, run:

```
    echo "source /opt/ros/melodic/setup.bash" >> ~/.bashrc
source ~/.bashrc
```

You can also open your *.bashrc* file to source it manually, up to you.
To install this tool and other dependencies for building ROS packages, run:

```
    sudo apt install python-rosdep python-rosinstall
    python-rosinstall-generator python-wstool build-essential
```

All this information can be found at `https://wiki.ros.org/melodic/Installation/Ubuntu`

6.3 Create a ROS Workspace

We assume that the installation was successful and we will proceed with the workspace set up which is crucial step, wrong set up may lead to disaster during development. Note that we use **CATKIN** in this project.
To create and build catkin workspace:

```
$ mkdir -p ~/mycatkin_ws/src
$ cd ~/catkin_ws/
$ catkin_make
```

6.4 Robot model installation

Navigate to

```
    https://github.com/unitreerobotics/unitree_ros
```

to get the repository required, then clone it to your **src** directory under your workspace.
It should look like this: Next, open the file unitree_gazebo/worlds/stairs.world under you directory and

```
prof@AS2M-HAOUAS5:~/mycatkin_ws/src$ ls
aws-robomaker-small-warehouse-world  realsense-ros
beginner_tutorials                   unitree_legged_sdk
CMakeLists.txt                       unitree_ros
realsense_gazebo_plugin              unitree_ros_to_real
```

include the below section to replace the one provided

```
<include>
<uri>model:/home/prof/mycatkin_ws/src/unitree_ros/unitree_gazebo/
   worlds/building_editor_models/stairs</uri>
</include>
```

This will change the path of building_editor_models/stairs to the one of the computer.
To build the entire packages, navigate to the workspace and build using catkin_make command:

```
cd ~/mycatkin_ws
catkin_make
```

error on building Warning, you will encounter an error of CMAKE, after some days of searching we found that the issue was addressed in one of the risen issue **Problem with building unitree_controller package 53**.

Below is the full pdf for solving the build issue.

```
https://github.com/unitreerobotics/unitree_ros/files/9492749/ROS.
   Build.instructions.for.Ubuntu.pdf
```

You can then run this commands to visualize the robot on gazebo and rviz for a starting point:
Description of robot A1

```
    roslaunch a1_description a1_rviz.launch
```

Launch the Gazebo simulation

```
    roslaunch unitree_gazebo normal.launch rname:=a1 wname:=stairs
```

The robot will be at rest position, to make the robot stand, run the servo node controller.

```
    rosrun unitree_controller unitree_servo
```

More commands can be found in then README.md file.

6.5 Controller implementation

The controller package can be found at :

```
    https://github.com/wessamhamid/unitree_simulation
```

Download the repository to the local repository update the build. The steps to follow are:

1. Change mode for all the ***.py** files and update the python3 to python2 to be executable.
2. Move files holding Robotcontroller lib, inversekinamtics, robotutilities and the python file for the controller to the src folder under the unitree_controller package.
3. Move the robot_controller.lauch to the launch folder.
4. Update CmakeLists by adding rospy, sensor_msgs to the catkin EQUIRED COMPONENTS

```
    find_package(catkin REQUIRED COMPONENTS
    controller_manager
    genmsg
    joint_state_controller
    robot_state_publisher
    roscpp
    gazebo_ros
    std_msgs
    tf
    geometry_msgs
    unitree_legged_msgs
    rospy
    sensor_msgs
)
```

And update the Package.xml to include the python build dependency

```
<build_depend>rospy</build_depend>
<build_export_depend>rospy</build_export_depend>
<exec_depend>rospy</exec_depend>
```

5. Navigate to the robot description and under config folder update the **.yaml** file to include the coefficients of the PID controller. An example:

```
    # FL Controllers ---------------------------------------
    FL_hip_controller:
    type: effort_controllers/JointPositionController
    joint: FL_hip_joint
    pid: {p: 105, i: 0.5, d: 4.5, i_clamp_min:  -1000,
       i_clamp_max: 1000}
```

6. Use of package teleop_twist_keyboard, on `http://wiki.ros.org/teleop_twist_keyboard` for more information.
   Update the main launch file **normal.launch** under unitree_gazebo package by including the customized teleop_twist_keyboard and the robot_controller like:

```
</include>
<!-- load NEW unitree_controller -->
<include file="$(find unitree_controller)/launch/robot_controller
   .launch"/>

<!-- New lines added for TWIST MUX-->
<include file="$(find quadruped_unitree)/launch/custom_twist_mux.
   launch"/>
```

7. Finally on different terminals run this below commands for full control of the robot. **Terminal 1**

```
roslaunch unitree_gazebo normal.launch rname:=a1 wname:=earth
```

Terminal 2

```
rosrun teleop_twist_keyboard teleop_twist_keyboard.py cmd_vel:=/
   key_vel
```

You can replace the wname by **space, stairs**.

6.6 Sensors Integration

To integrate an external sensing device on the robot, it is important to get the meshes files , it could be in **.dae** or **.stl** format. The procedure will be similar for the sensors.

Intel Realsense d435

```
prof@AS2M-HAOUAS5:~/mycatkin_ws/src/realsense-ros/realsense2_description$ tree
.
├── CHANGELOG.rst
├── CMakeLists.txt
├── launch
│   ├── gazebo.launch
│   ├── view_d415_model.launch
│   ├── view_d435i_model.launch
│   ├── view_d435_model.launch
│   ├── view_d435_model_rviz_gazebo.launch
│   ├── view_d455_model.launch
│   ├── view_l515_model.launch
│   ├── view_multiple_d435_models.launch
│   ├── view_r410_model.launch
│   └── view_r430_model.launch
├── meshes
│   ├── d415.stl
│   ├── d435.dae
│   ├── d455.stl
│   ├── l515.dae
│   ├── plug_collision.stl
│   └── plug.stl
├── package.xml
├── rviz
│   └── urdf.rviz
├── tests
│   ├── dual_d415.xacro
│   ├── dual_d435.xacro
│   ├── dual_r410.xacro
│   ├── dual_r430.xacro
│   ├── one_of_each.xacro
│   └── test_xacro.py
└── urdf
    ├── _d415.urdf.xacro
    ├── _d435.gazebo.xacro
    ├── _d435i_imu_modules.urdf.xacro
    ├── _d435i.urdf.xacro
    ├── _d435.urdf.xacro
    ├── _d455.urdf.xacro
    ├── _l515.urdf.xacro
    ├── _materials.urdf.xacro
    ├── _r410.urdf.xacro
    ├── _r430.urdf.xacro
    ├── test_d415_camera.urdf.xacro
    ├── test_d435_camera.urdf.xacro
    ├── test_d435i_camera.urdf.xacro
    ├── test_d435_multiple_cameras.urdf.xacro
    ├── test_d455_camera.urdf.xacro
    ├── test_l515_camera.urdf.xacro
    ├── test_r410_camera.urdf.xacro
    ├── test_r430_camera.urdf.xacro
    └── _usb_plug.urdf.xacro

5 directories, 45 files
```

1. Open a new terminal and navigate to **src** directory and there clone the original repository from Intel at : IntelRealSense /realsense-rosPublic Here we are only interested in the gazebo plugins, xacro and meshes files pf the d435 only in our case. The tree of the project is as follows:

2. Create a subdirectory under the robot description package and include the extract files as shown in the tree below:

```
prof@AS2M-HAOUAS5:~/mycatkin_ws/src/unitree_ros/robots/a1_description$ tree
.
├── CMakeLists.txt
├── config
│   ├── controllers.yaml
│   └── robot_control.yaml
├── launch
│   ├── a1_rviz.launch
│   └── check_joint.rviz
├── meshes
│   ├── calf.dae
│   ├── d435.dae
│   ├── hip.dae
│   ├── lds.stl
│   ├── thigh.dae
│   ├── thigh_mirror.dae
│   ├── trunk_A1.png
│   └── trunk.dae
├── package.xml
├── urdf
│   └── a1.urdf
└── xacro
    ├── const.xacro
    ├── gazebo.xacro
    ├── includes
    │   ├── gazebo
    │   │   ├── _d435.gazebo.xacro
    │   │   └── laser.gazebo.xacro
    │   └── parts
    │       └── sensors
    │           ├── laser
    │           │   └── laser.urdf.xacro
    │           └── realsense
    │               └── _d435.urdf.xacro
    ├── includes.xacro
    ├── leg.xacro
    ├── materials.xacro
    ├── robot.xacro
    ├── stairs.xacro
    └── transmission.xacro
```

3. Update the robot xacro file to include the external component

```
<!-- Xacro arguments for depth camera use -->
<xacro:arg name="realsense_enable" default="true" />
<xacro:arg name="pointcloud_enable" default="true" />
    <!-- SENSORS -->
<!-- REALSENSE CAMERA -->
<xacro:if value="$(arg realsense_enable)">
<xacro:sensor_d435 name="realsense_d435" topics_ns="
   realsense_d435" parent="trunk" publish_pointcloud="$(arg
   pointcloud_enable)" >
<origin xyz="0.27 0 0" rpy="0 0 0" />
</xacro:sensor_d435>
</xacro:if>
```

Note that the camera node has **trunk** as parent and was placed at **0.27 cm** from **x direction**. this value is deduced from the values provided in **const.xacro** according to trunk dimensions by Unitree Robotics.

LiDAR The sensor used was from a part of project of **Rangel Isaias Alvarado Walles** on jetbot_diff_drive
Similar to the previous sensor:

1. Open a new terminal and navigate to **src** directory and there include the LiDAR meshes and xacro file downloaded from the repository jetbot_description and include to the subdirectory created for the LiDAR shown in the tree of robot description packages similar to the camera process
2. Update the robot xacro file to include the external component

```
<!-- Xacro arguments for depth lidar use -->
<xacro:arg name="lidar_enable" default="true" />
<xacro:arg name="visualize_laser" default="true" />
<!-- RPLIDAR LASER 0.058-->
<xacro:if value="$(arg lidar_enable)" >
  <xacro:sensor_rplidar name="rplidar" topic_ns="scan" parent="
     trunk" visualize_laser="$(arg visualize_laser)" >
    <origin xyz="0 0 0.07" rpy="0 0 0" />
  </xacro:sensor_rplidar >
</xacro:if>
```

Note that the camera node has **trunk** as parent and was placed at **0.07 cm** from **Z direction**. This value is deduced from the values provided in **const.xacro** according to trunk dimensions by Unitree Robotics.

New world for motion planning and obstacle avoidance steps to follow:

1. Navigate to the src directory and clone the required GitHub repository from **AWS RoboMaker**

```
git clone https://github.com/aws-robotics/aws-robomaker-small-
   warehouse-world.git
```

2. Move the model and worlds subdirectory to the world directory of the unitree_gazebo package

6.7 Implementation of navigation stack on virtual robot

In this section we will discuss the process to setup the navigation stack with all the dependencies to get the robot performing autonomous navigation within an environment from a start position to a desired position by creating an optimum path. To achieve it this will require the following ros package:

- Hector_slam
- AMCL
- Move_base

6.7.1 Mapping and Localization (SLAM)

Befort planning to proceed with the navigation stack it is crucial to generate a map. Through map, the robot can accurately determine its specific position inside en environment, therefore the robot can compare sensors data against landmarks. In our case we will use hector_slam a ros1 package for simultaneous localization and mapping.

There are two possibilities to get the package, either install it with the apt-get or clone the repository to our work space and built it. The last is the the method used in our case, this will give the flexibility to develop our software and configure it to fit our need. Let's break down the process.

Open a new terminal and run the command

```
sudo apt install ros-melodic-gmapping
```

then navigate to the working space, once there run the command below to clone the repository:

```
git clone https://github.com/tu-darmstadt-ros-pkg/hector_slam.git
```

The above command will clone the repository in a sub-directory named **hector_slam**, then navigate back to the ws (work space) and build to generate the executable files.

In the hector_slam sub-directory we are interested in :

- hector_mapping which is the SLAM node.
- hector_geotiff for saving of map and robot trajectory to geotiff images files.
- hector_trajectory_server for saving tf based trajectories.

These are launch at once in a single launch file that we named **slam.launch**

Further configuration needs to be done to get it work properly , for this navigate to the hector_slam subdirectory and edit **geotiff_mapper.launch** and change the **trajectory-source-frame-name** to our own which is **/base_link**.

similarly in **mapping_default.launch** change the **base-frame** to **trunk** and **odom_frame** to **base_link**. There is no need to change the scan topic because the topic name matches with the standard. The map size and update parameters could be tuned depending on the performance evolution.

To test that all is well configured, navigate to the ws, build and run

```
roslaunch unitree_gazebo slam.launch
```

You can now launch your robot and start moving around to draw the map, once satisfied you can save your map using

```
rosrun map_server map_saver -f ~/home/my_map
```

6.7.2 Navigation

There comes the navigation stack, first of all you will need to install package and for this open a new terminal and run the commands below:

```
sudo apt-get install ros-melodic-navigation
```

In addition some dependencies:

```
sudo apt-get install ros-$ROS_DISTRO-move-base
sudo apt-get install ros-$ROS_DISTRO-actionlib
sudo apt-get install ros-$ROS_DISTRO-geometry-msgs
sudo apt-get install ros-$ROS_DISTRO-std-msgs
```

Again the above commands are not necessary since we have installed ros-desktop that includes all the possibles dependencies.

They can be found in **cd /opt/ros/melodic/share** and there you can search for all the installed packages. We will make use of the **Turtlebot3-navigation** to speed up our work.

Keep in my mind that its a differential drive in the case of turtlebot so the parameters are not the same for our robot with specific drive, therefore omni-directional drive will be used.

In order to make use of the package, copy the folders of launch, param to unitree_gazebo subdirectory and let's make significant changes to adapt to our legged robot.

In **/home/prof/mycatkin_ws/src/unitree_ros/unitree_gazebo/params** would have

```
prof@AS2M-HAOUAS5:~/mycatkin_ws/src/unitree_ros/unitree_gazebo/
   params$ ls
costmap_common_params.yaml      global_costmap_params.yaml
DBlocal_planner_params.yaml     local_costmap_params.yaml
dwa_local_planner_params.yaml   nav_obstacles_params.yaml
```

which are all the yaml files containing the parameters required for the **move_base** node. We inspired this from the real robot yaml configuration.

And in **/home/prof/mycatkin_ws/src/unitree_ros/unitree_gazebo/launch** you have

```
prof@AS2M-HAOUAS5:~/mycatkin_ws/src/unitree_ros/unitree_gazebo/
   launch$ ls
a1_rviz.launch  custom_twist_mux.launch  nav.launch     slam.launch
amcl.launch     move_base.launch         normal.launch  tf.launch
```

which hold the launch files required, we made this single folder to help avoid any confusion.

In **amcl.launch** was updated to odom model type, frame ID and base frame ID were configured to our robot configuration.

In **move_base.launch** we have included the necessary yaml files and the argument for velocity and odom topic.

NB: the tricky part is the update of parameters to achieve high performance in optimum path generation and optimum navigation.

Once all these configure were done , we can now proceed to check the working of the overall.

Open multiple terminals and navigate to your ws then run the commands below in different terminals

```
roscore
roslaunch unitree_gazebo normal.launch rname:=a1 wname:=
   no_roof_small_warehouse
roslaunch unitree_gazebo slam.launch
roslaunch unitree_gazebo amcl.launch
roslaunch unitree_gazebo move_base.launch
```

The first command start the master which will create connection between nodes by assigning for each node a specific ID so that there is no confusion during communication phase.

The second command launch your robot on gazebo environment and open rviz , so far so you can get the output perception on rviz and can display the topic list, info and echo from a new terminal.

The third command launch the slam node , this node subscribe to the topic scan and publish in return a map topic.

The fourth command launches the adaptive Mont Carlo node that will generate the path by localizing potential obstacle, in other word node uses the particle positions to compute and publish the transform from map to base_link and finally the fifth command launches the move base node which is responsible for sending nav_velocity.

Few topics info are listed below to observe the publishers and subscribers to each one.

```
prof@AS2M-HAOUAS5:~/mycatkin_ws$ rostopic info /scan
Type: sensor_msgs/LaserScan
Publishers:
 * /gazebo (http://AS2M-HAOUAS5:34745/)
Subscribers:
 * /rviz (http://AS2M-HAOUAS5:40873/)
 * /hector_mapping (http://AS2M-HAOUAS5:43703/)
 * /amcl (http://AS2M-HAOUAS5:37015/)
 * /move_base (http://AS2M-HAOUAS5:42909/)
```

Listing 1: scan topic info

```
prof@AS2M-HAOUAS5:~/mycatkin_ws$ rostopic info /map
Type: nav_msgs/OccupancyGrid
Publishers:
 * /hector_mapping (http://AS2M-HAOUAS5:43703/)
Subscribers:
 * /rviz (http://AS2M-HAOUAS5:40873/)
 * /move_base (http://AS2M-HAOUAS5:42909/)
```

Listing 2: map topic info

```
prof@AS2M-HAOUAS5:~/mycatkin_ws$ rostopic info /twist_mux/cmd_vel
Type: geometry_msgs/Twist
Publishers:
 * /twist_mux (http://AS2M-HAOUAS5:34651/)
 * /move_base (http://AS2M-HAOUAS5:42909/)
scribers:
 * /twist_marker (http://AS2M-HAOUAS5:36815/)
 * /ROBOT_CONTROLLER (http://AS2M-HAOUAS5:45409/)
```

Listing 3: cmd_vel topic info

```
prof@AS2M-HAOUAS5:~/mycatkin_ws$ rostopic info /odom
Type: nav_msgs/Odometry
Publishers:
 * /gazebo (http://AS2M-HAOUAS5:34745/)
Subscribers:
 * /move_base (http://AS2M-HAOUAS5:42909/)
```

Listing 4: odom topic info

7 Annexes B: Real robot

7.1 Overview

This section will introduce a step by step process to set up remote connection between the host PC and the hardware and perform SLAM and 2D camera visualization.

7.2 Network configuration

First of all one should make sure that the WiFi on both the host PC and robot can connect to each other, this was done by the help of the network technician from AS2M department. Meaning that host PC and robot can communicate remotely.

Secure Shell (SSH)is a cryptographic network protocol for operating network services securely over an unsecured network furthermore It is a software package that enables secure system administration and file transfers over insecure networks. And the good news is that Most Linux distributions come with SSH client installed by default with is our case. In order to access the TX2 board from the host PC open a new terminal and run

```
ssh unitree@192.168.123.12
```

Then you type the password: `123`

If everything went well you should log in to the the TX2 board and can start developing and managing tasks.

A quick reminder, the robot has ROS Melodic installed and a Package for SLAM (Simultaneous Localization And Mapping) under the workspace. Thus, every other package will be added so that the development can be under the same workspace for all.

7.3 setting up a remote roscore

It's essential to configure the IP addresses correctly in order to visualize the data of published topics from the multiple remote nodes Through rviz from the host PC. Configure as follows:

Navigate to home directory and edit *bahsr* file with preferred editor (vim, nano, gedit...), this should be done in both robot and host PC.

Remember the robot is the master sending topics and the host PC receives.

The variable ROS_IP tells ROS nodes which IP:PORT combo they can be reached at, and the variable ROS_MASTER_URI specifies the IP:PORT combo at which the master (roscore) can be reached. On the robot's *bahsr* file add :

```
    ROS_MASTER_URI=http://192.168.132.12:11311
    ROS_IP=192.168.132.12
```

And similarly on the host PC add:

```
    ROS_MASTER_URI=http://192.168.132.12:11311
    ROS_IP=0.0.0.0
```

In addition edit on both /etc/hosts: file and add the IP address and hostname of each in each Incase, if your hostname isn't configured with DNS (common with laptops), it's essential to configure it to use the

correct IP address, but not necessarily if everything works well. To confirm open a new terminal log in to the robot using ssh and run

```
roscore
```

In another terminal on your host PC run

```
rostopic list
```

You should see the topics listed then run

```
rostopic echo \name_of_topic
```

to see the data being published.

7.4 Setting Up the RPLIDAR

A quick recap RPLIDAR A1 is a 2D LIDAR sensor that can scan the environment in 360° and produce a map of the area.

The manufacturer provides us with a Ethernet and USB Cables to connect the rplidar to the robot development board and power source. The connection is shown in the Figure 19 below :

Figure 19: Connection Lidar on robot

The slam package was provided. Assuming the executables were generated if not navigate to the workspace

```
cd ~/catkin_ws
```

And build to generate all the executable files using

```
catkin_make
```

To perform slam, log in to the TX2 board,open 2 new terminals and run consecutively:

```
Terminal 1: roscore
Terminal 2: roslaunch slam_planner slam_planner_online.launch
```

NB: The *slam_planner_online.launch* has its dependency which is

slam_planner_rplidar_start.launch and the last has its dependencies:

slam_planner_sdk.launch This helps to avoid confusion and lengthy launch file. To comprehend more navigate to

```
cd ~/catkin_ws/src/slam_planner
```

And explore for further understanding.

Open a new terminal in your host PC and run

```
rosrun rviz rviz
```

This command will bring up rviz and from the graphical interface add by new topics : map, scan, tf. Finally change the fixed frame to map it was not the default one. You should see the map drawing has the robot moves.

To save the map for further usage, run one a new terminal

```
rosrun map_server map_saver -f name_of_the_map
```

The command will create two files, map.pgm and map.yaml. The map in .pgm format (the portable gray card format) and the map configuration file .yaml.

7.5 Setting Up the Camera

Quick reminder The Intel RealSense SDK for Linux provides libraries, tools, and samples to develop applications using Intel RealSense cameras, over the Intel librealsense API To begin, it's import to confirm that librealsense has been installed and working, for this log in to the TX2 and run

```
realsense-viewer
```

If the command fails, you will need to install it from the guidelines from Intel realsense. Next, you should install the ROS package for realsense from ros1-legacy specific for ROS1 by running:

```
sudo apt-get install ros-$ROS_DISTRO-realsense2-camera
```

The package comes with various launch files but we are specifically interested in the *rs_camera.launch* at this stage.

This package was not provided by the manufacturer hence some additional modifications need to be done. This concerns the tf transformation and fixed frame conflicts. If you've explored the provided launch files in the slam_planner package you will notice this below transformations:

It is import to note that each frame must be connected directly or indirectly to each other in the tree structure and since we are using `map` as fixed frame we should transform the `camera_link` so that they are related. To do so edit the *slam_planner_rplidar_start.launch* file and include this line of code to the Static Transform Publishers tag:

```
<node name="base2camera" pkg="tf" type="static_transform_publisher"
      args="0 0 0 0 0 1 /base_link /camera_link 100" />
```

Once this is done and saved. Login to the TX2 and open 3 terminals and run consecutively:

```
roscore
roslaunch slam_planner slam_planner_online.launch
roslaunch realsense2_camera rs_camera.launch
```

In case you want to used the RGBD launch file then first install

```
    sudo apt-get install ros-melodic-rgbd-launch
```

then run

```
    roslaunch realsense2_camera rs_rgbd.launch
```

Open a new terminal on your host PC and check the topics

```
    rostopic list
    rostopic info /topic_name
    rostopic echo /topic_name
    rostopic type /topic_name
```

Then Finally open a new terminal and launch rviz

```
    rosrun rviz rviz
```

and from the graphical interface add from topics, tf, map, scan and camera_raw then select map as fixed frame. And to visualize the tree structure of the static frames transformation :

```
rosrun rqt_tf_tree rqt_tf_tree
```