POLITECNICO DI TORINO

MS's Degree in Electronic Engineering

Politecnico
di Torino

MS's Degree Thesis

Deterministic access to mobile edge for robot control

Supervisors

Prof. Carla Fabiana CHIASSERINI

Prof. Corrado PULIGHEDDU

Candidate

Davide BOGGIO MARZET

Academic Year 2023-2024

Summary

In modern industrial environments, the need for automated solutions increased as mobile network technology advanced. With the deployment of fifth-generation standards (i.e., 5G), it is now possible to achieve a level of automation never before reached. 5G allows low latency, high throughput, and large bandwidth, making it possible to deploy time-sensitive applications such as autonomous driving and automated robot control in an industrial environment. This thesis focuses on the deployment of a low-cost 5G network exploiting the tools provided by the open-source software OpenAirInterface (OAI) and the successive development of a Robot Operating System (ROS) 2 mobile robot application that could be deployed across the 5G network to automatically control the robot's movement. Then, the performances of the system were studied, and the main criticalities were analyzed. Performance metrics, particularly latency, were analyzed as a critical factor in an automated application to ensure the safety of the robot itself and human workers. Different implementations of the robot application were tested, and the best possible solution was proposed for the use case under investigation. The results demonstrate that reliable robot control can be achieved over a 5G network, meeting low-latency requirements. Future research could explore more agile robots, complex control laws, and commercial 5G systems with enhanced channel conditions and network slicing for increased reliability.

Table of Contents

List of Tables

List of Figures

Acronyms

UE User Equipment

RAN Radio Access Network

CN Core Network

gNB g Node B

NF Network Function

SDR Software Defined Radio

MEC Multi-access Edge Computing

OAI Open Air Interface

MEP Multi-access Edge computing Platforms

KPI Key Performance Indicator

ROS Robot Operating System
RTT Round Trip Time

BW Bandwidth

SNR Signal to Noise Ratio

BLER Block Error Rate

PRB Physical Resource Block

Chapter 1

Introduction

1.1 Thesis aim

This thesis aims to evaluate the behavior, performance, and sensitivity of a 5G network optimized with local EDGE data processing in a typical Industry 4.0 use case: the remote control of a mobile robot, particularly in response to changes in key system parameters.
5G characteristics, in particular low-latency, make possible to deploy time-sensitive applications; therefore in this thesis will be investigated how the control of a robot performs when deployed across the network.
Since this work is based on internationally defined standards, the test environment has been created by exploiting open-source tools.
The key steps of this work include establishing a 5G connection between a gNB and a UE using OAI tools, developing a ROS 2 application, and deploying it across the established 5G network. Finally, the system's performance will be assessed, and the collected data will be analyzed and correlated with the RTT plot.

1.2 Thesis structure

This thesis work is organized as follow:

- **Chapter 1**:
  Introduction of the thesis objectives and a description of its organizational structure.

- **Chapter 2**:
  The second chapter presents an overview of the 5G technology and the reasons why it has become so relevant in today's low-latency, high-throughput applications. It also introduces OAI, which is the main tool exploited in this

thesis work and presents the network setup phase and the main results of the connectivity tests.

- **Chapter 3**:
  The third chapter presents MEC and OAI MEP, underlining the architecture and the potential benefits. It discusses the implementation of such infrastructure and its limits.

- **Chapter 4**:
  The fourth chapter introduces the framework employed to control the robot, ROS2. It offers an overview of the robot's application development and successive deployment, showing the main results of these phases.

- **Chapter 5**:
  The fifth chapter focuses on the deployment of the application across the OAI 5G network, presenting the setup phase and the several tests performed to assess the system's performance. Different sections focus on specific sets of experiments carried out to study the effects that the various parameters have on the system.

- **Chapter 6**:
  The last chapter summarizes the main results and discusses the issues raised during this thesis work, analyzing the main criticality and the possible future developments.

Chapter 2

5G technology

2.1 Overview

5G is the fifth generation set of standards for mobile networks. It is defined by the 3GPP alliance, and therefore can work either in a intra-vendor and extra-vendor environment.
The main objectives of 5G are to fit market traffic growth, improve user experience and industrial applications capabilities. In order to achieve the latter is essential to provide low latency and fast response. With respect to the previous generations, 5G exploit new technologies such as Slicing, EDGE computing, Network Function Virtualization, Non Terrestrial Network, etc, which allows to improves the 4G platform in different aspects: flexibility, reliability, latency, data-rates and traffic density. Thanks to this new features and improvements, 5G can be employed in new areas like autonomous driving, critical industrial applications, automation, smart cities and so on.
An additional characteristic is the compatibility between 5G and 4G technologies, allowing a smooth transition towards the latest generation of standards.

2.2 5G architecture

5G can be deployed in two configurations: Stand-Alone (SA) or Non-Stand Alone (NSA). The SA configuration relies only on fifth-generation RAN (i.e. gNB), while the NSA exploits also 4G RAN and CN. The latter can be seen as a temporary implementation while waiting for a full 5G network deployment, allowing a smooth 5G introduction while relying on the 4G core network, exploiting 5G as a booster for 4G. For this reason, only the 5G SA configuration is considered in this work.
The 5G architecture can be schematically represented as composed of three main elements: the UE, the RAN and the CN, as shown in figure 2.1.

Figure 2.1: 5G architecture [1]

The UE is any device used by an end-user, human or not, to access the network, while the RAN provides connectivity between the user and the Core Network. The 5GC (5G core network) is a Service Based Architecture that exploits different Network Functions (NF) to provide services. The different NFs are connected through interfaces in a common framework as shown by figure 2.2.

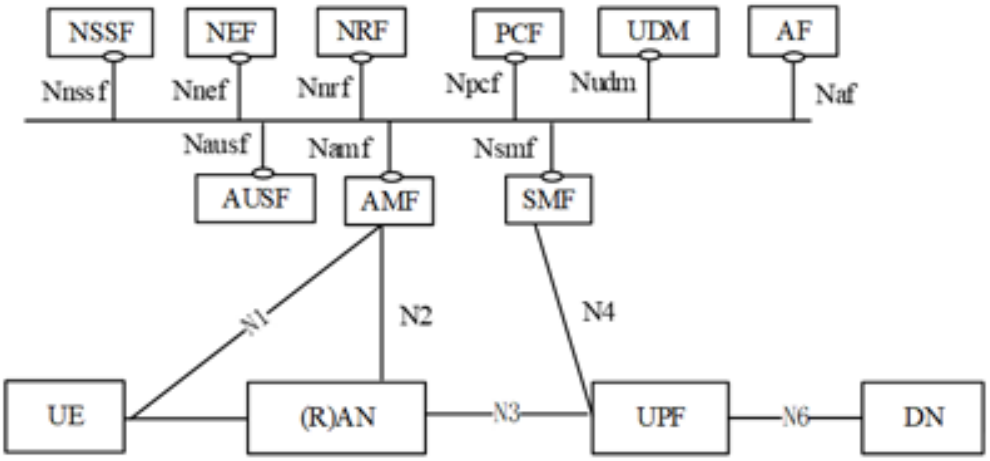


Figure 2.2: Main NFs in 5G [1]

Three of the most important NFs are:

- UPF: the User Plane Function performs all user plane functions on the packets, such as forwarding, routing, marking, QoS, inspection, and so on; it is the key element for latency performance.
- AMF: Access and Mobility management Function, in charge of registration, reachability, and connection. It also performs mobility management, UE authentication and authorization.

- SMF: Session Management Function handles selection and control of user plane functions, and manages the UE IP address.

The others NFs represented in figure 2.2 are employed in the Control Plane (upper part of the figure) and in the User Plane (bottom part):

- Data Network (DN): this is primarily involved in the User Plane, which means it handles the actual data transfer and internet traffic for users. It represents the external networks like the internet or other service networks that the user accesses.
- Application Function (AF): manages and controls specific applications, and it may also interact with the User Plane.
- Unified Data Management (UDM): manages user subscription information and authenticates users and authorizes their access to the network.
- Policy Control Function (PCF): ensures that user data traffic adheres to the policies and bearer capacities. This means it manages quality of service and enforces rules related to data usage, making sure that traffic does not exceed the limit.
- Network Repository Function (NRF): keeps track of all other Network Functions (NFs) by registering, deregistering, and updating their statuses and services. It acts like a directory that helps different NFs find and communicate with each other.
- Network Exposure Function (NEF): exposes network capabilities to external applications securely.
- Authentication Server Function (AUSF): handles user authentication.
- Security Anchor Functionality (SEAF): works with AUSF to ensure secure access and communication.
- Network Slice Selection Function (NSSF): responsible for selecting the appropriate network slice for a given service or user.

A key feature of the CN is the decoupling of control and data planes, allowing scalability, flexibility, and the possibility to deploy user plane closer to the end-user. A fundamental consequence is the Virtualization, which is the decoupling of the NFs from the physical equipment.

An important innovation of 5G with respect to previous standards is Edge computing. This technique moves part of the computational resources, including user

plane functionalities, as close as possible to the mobile devices, allowing lower latency and higher security as data are kept at the edge of the network. These are fundamental aspects of some applications like autonomous driving, gaming, or near-real-time applications.

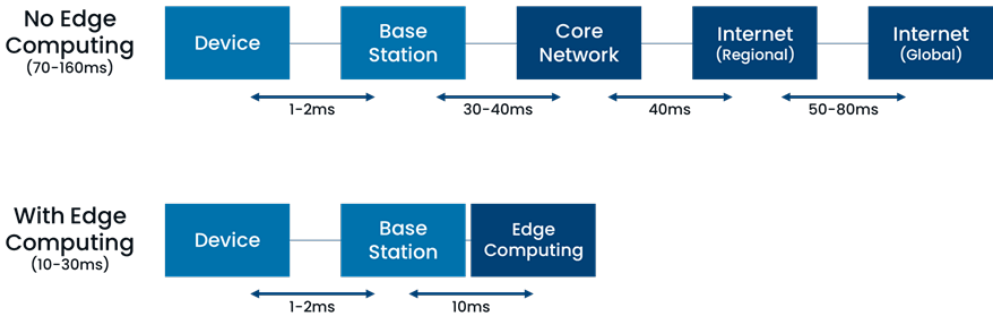


Figure 2.3: Latency with and without Edge Computing [2]

Network slicing allows the 5G network to be divided into multiple virtual networks, each optimized for a specific type of service or user need and dynamically used only for the time service is requested. This ensures efficient and customized network performance, providing a better quality of service.

2.3 Open Air Interface

OpenAirInterface Software Alliance (OSA) is a non-profit consortium founded by EURECOM that aims to develop an open-source environment for 5G development. OSA provides all the repositories and codes required to deploy a basic 5G environment, made of CN, gNB, and UE. It also provides different projects that are currently work-in-progress, like the MEC platform, Network slicing, and others.

2.4 Private 5G network

For the thesis work, a Stand-Alone architecture was set up using two USRP B210 devices, one for the CN/gNB and the other for the UE, that serve as transmitter and receiver. These devices [3] are compatible with the OAI network and are SDRs, meaning that traditional analog components are implemented in software or embedded systems, making them ideal for low-cost experimentation.
In order to program and control such devices, two PCs were needed.

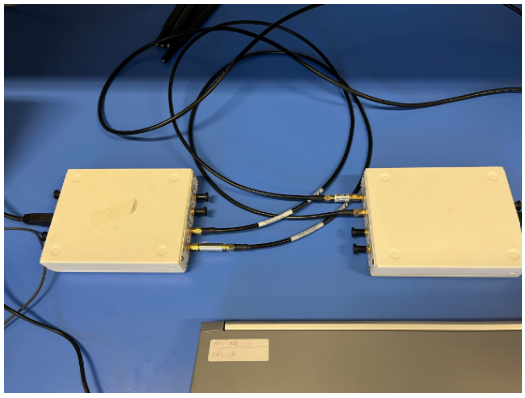


Figure 2.4: USRP B210 devices

The minimum hardware requirements reported on the OAI tutorial[4] for the two PCs are:

1. PC hosting gNB/CN:
   - OS: Ubuntu 22.04 LTS
   - CPU: 8 cores x86-64 @ 3.5 GHz
   - RAM: 32 GB
2. PC hosting UE:
   - OS: Ubuntu 22.04 LTS
   - CPU: 8 cores x86-64 @ 3.5 GHz
   - RAM: 8 GB

Politecnico di Torino provided the necessary PCs because my laptop did not meet these requirements. The laboratory's laptops didn't fully satisfy the requirements as the gNB's RAM is 16 GB, and both PCs have a CPU clock smaller than the required one. These limitations do not preclude the working of the system but might limit the performance as seen in the following chapters.

The first procedure to be done consisted in enabling the two USRP to communicate with each other either with a cabled and a wireless connection. For this purpose, Docker containers and OAI repositories were employed.
The wired connection relies on two cables to connect the gNB and the UE with the addition of a 30 dB attenuator per cable to simulate a more realistic channel; while the wireless connection exploits two antennas per device. The wired connection provides stronger connection as less attenuation is achieved, but is not representative

of a real-life scenario as mobile networks relies on wireless connections, therefore was just to test the connection in the best possible and stable condition.
Once the CN and the gNB were up and running, it was possible to detect the presence of a signal emitted by the gNB exploiting a simple GNU Radio program, as visible from the figures 2.5 and 2.6. Such signal was emitted within the band 41 range and was correctly detected.
GNU Radio is an open-source software that, combined with the B210, allows the creation of SDR exploiting signal processing blocks.

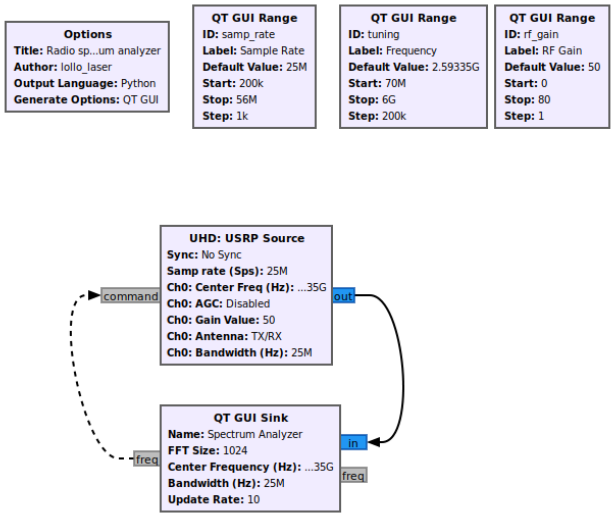


Figure 2.5: GNU Radio program

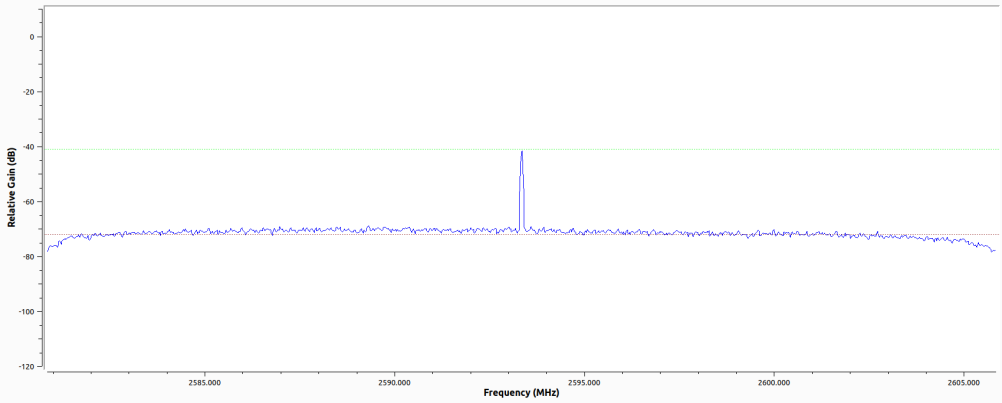


Figure 2.6: Detected peak in band 41

After this verification, it was possible to connect a COTS UE (i.e., a smartphone)

to the gNB. Unfortunately, OAI doesn't support the latest version of Android installed in the COTS UE, causing the connection to last just a few seconds. Still, this test allowed to verify the correct functionality of the gNB.
The following step involved the connection of the nrUE (i.e., the other USRP) to the gNB. This procedure could be performed by setting both devices to work at the same frequency using two commands given via the Ubuntu terminal.
Once the two devices were connected, several tests could be performed, including throughput and latency tests, which are particularly useful for evaluation purposes.

Band	Wired	Wireless
n41	17.3	14.7
n78	20.2	14.6

Table 2.1: Average throughput expressed in Mb/s

To perform the throughput test was employed iPerf, which is an open-source software that allows to measure throughput by creating a stream of data between a server and a client with target bandwidth of 100 Mbps.
As a first step, it is necessary to enter the oai-ext-dn Docker container from the gNB's PC; this passage is essential as the container is isolated and therefore would be impossible to perform the test without accessing it. Then, it is necessary to create a client on the gNB and a server on the other machine.
As expected, the wired connection provided higher throughput as better channel conditions were achieved, as reported in table 2.1.
The second test to be performed is the latency test, which could be easily done by pinging the UE IP address from the oai-ext-dn container. These measurements provide the round-trip time, which is approximately twice the latency plus a small overhead due to processing. The results can be seen in Figure 2.7, where it is evident that a wired connection at a lower frequency band shows similar performance to a wireless connection in Band 78.

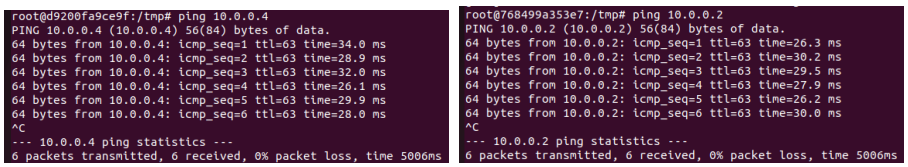


Figure 2.7: Ping test in wired Band 41 (left) and wireless Band 78 (right) cases

A major problem was the general instability of the system, it was quite common

for the connection to suddenly drop, forcing the OAI system to be rebooted. This issue does not directly influence the results but translates into more time required to complete an experiment.

Chapter 3

Multi-access Edge computing Platform

3.1 MEC overview

Multi-access Edge Computing (MEC) enables MEC applications to be implemented as software-only entities operating on a virtualization infrastructure located at or near the network edge. This allows for the execution of applications in an environment characterized by low latency, close proximity, and high bandwidth, with access to location and real-time radio network data. The radio conditions data are communicated through the MEC platform via the Radio Network Information Service (RNIS).
The Radio Network Information Service offers radio network related data to MEC applications and platforms.
The references for MEC and RNIS are provided by the European Telecommunications Standards Institute (ETSI).

3.2 OAI MEP

MEC applications interact with the MEP using the mp1 interface, while applications hosted at the MEP communicate with the Radio Access Network (RAN) and Core Network (CN) components through the mp2 interface. The MEC RNIS gathers radio network information via the mp2 interface and makes it available to interested applications through the mp1 interface.
A scheme of this architecture can be seen in figure 3.1.
The purpose of OAI MEP is to retrieve RAN KPIs and expose them to the MEP user. The OAI tutorial refers to a simulated environment, meaning that the

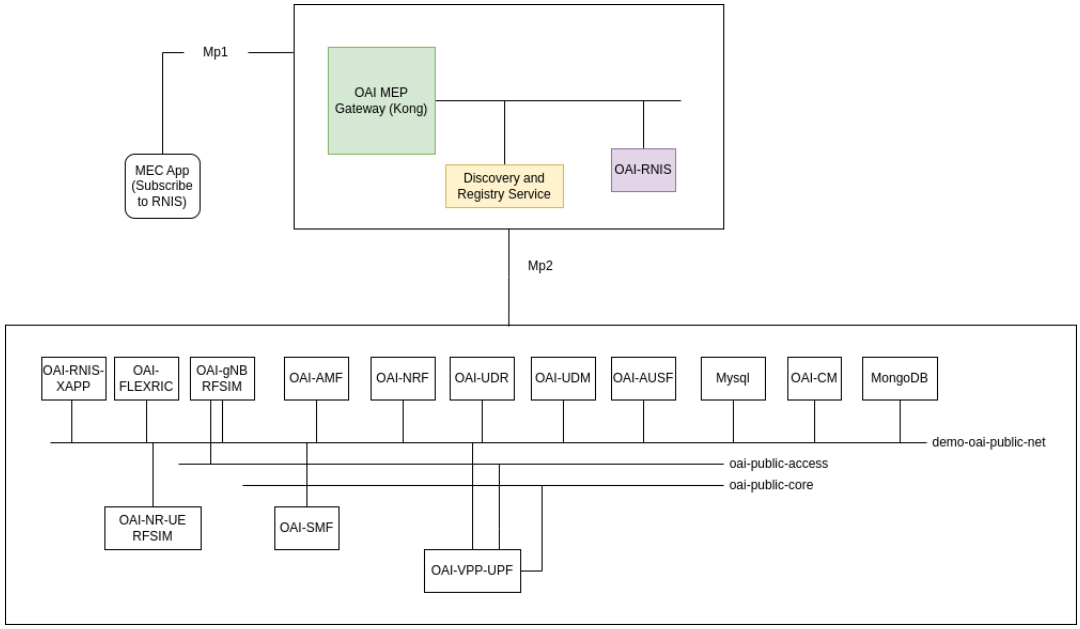


Figure 3.1: MEP structure [5]

reported KPIs are static and not referred to the real channel.
The MEP services provided by this application are reported in figure 3.2


```
[
  {
    "description": "The service allows other app or services to consume network and radio information",
    "endpoints": [
      {
        "method": "GET",
        "name": "rab_info",
        "path": "/queries/rab_info"
      },
      {
        "method": "GET",
        "name": "plmn_info",
        "path": "/v2/queries/plmn_info"
      },
      {
        "method": "GET",
        "name": "s1_bearer_info",
        "path": "/v2/queries/s1_bearer_info"
      },
      {
        "method": "GET",
        "name": "layer2_meas",
        "path": "/queries/layer2_meas"
      },
      {
        "method": "GET",
        "name": "get_subscriptions",
        "path": "/subscription"
      },
      {
        "method": "POST",
        "name": "post_subscriptions",
        "path": "/subscription"
      },
      {
        "method": "GET",
        "name": "get_subcription",
        "path": "/subscriptions/{subscriptionId}"
      },
      {
        "method": "PUT",
        "name": "modify_subscription",
        "path": "/subscriptions/{subscriptionId}"
      },
      {
        "method": "DELETE",
        "name": "delete_subscription",
        "path": "/subscriptions/{subscriptionId}"
      }
    ],
    "host": "oai-mep.org",
    "name": "rnis",
    "path": "/rnis/v2",
```


Figure 3.2: MEP services

3.3 Deployment of OAI MEP on real hardware

A first deployment attempt was made in a simulated environment, as suggested by the tutorial, and the procedure successfully produced the required KPIs as reported in figure 3.3


```
"rsrp": {
  "kpi": "rsrp",
  "labels": {
    "amf_ue_ngap_id": 1
  },
  "source": "RAN",
  "timestamp": 1711123479598819,
  "unit": "dBm",
  "value": -44
},
"snr": {
  "kpi": "snr",
  "labels": {
    "amf_ue_ngap_id": 1
  },
  "source": "RAN",
  "timestamp": 1711123479588821,
  "unit": "dBm",
  "value": 55.0
}
```


Figure 3.3: KPIs in a simulated environment

The most impactful problem of this procedure concerned Docker: sometimes, one of the deployed containers didn't start correctly or could get down while performing the setup operations. These issues led to the system being taken down and restarted all over, causing a huge waste of time.

To test the real channel conditions, the MEC application had to be deployed on the actual laboratory setup.
When deploying MEP on the two USRPs, some significant differences have to be applied to the configuration files provided by OAI regarding the UE and the gNB (see Appendix A.1):

- Added configuration file for the gNB
- Changed some flags to enable the use of real hardware
- Substituted serial address of the USRPs
- Added additional options like band and frequency

Once all the changes were made, it was possible to test the connection between the two devices.
Although the changes were effective and a connection was established between the UE and the gNB, the connection was not complete because it was not possible to ping each other. An example of this partial connection can be seen in figure 3.4.

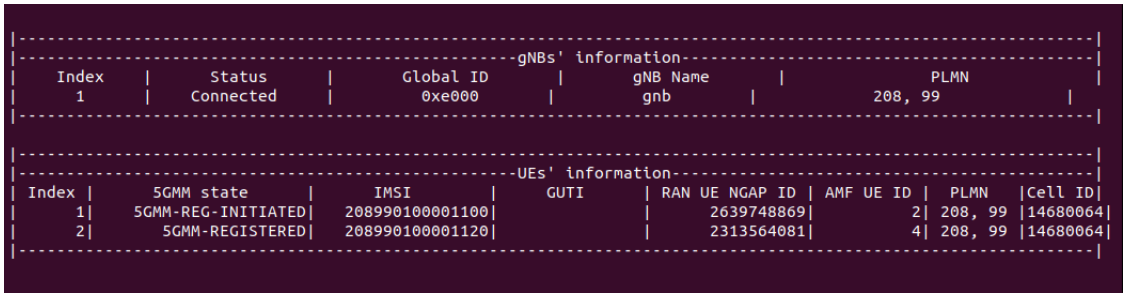


Figure 3.4: OAI-amf log

Despite this issue, it was still possible to perform some tests to verify the channel condition. In particular, SNR measurements were taken to characterize the channel. Even when deployed on real devices, the Docker containers suffered from the same issues reported above, causing the setup to be tried different times before actually working properly.
The characterization was performed at difference distances between UE and gNB to better understand the impact of distance on the channel. As figure 3.5 shows, the SNR clearly decreased while moving away the UE from the gNB.
The positions reported correspond to:

- Position 1: USRPs on the same table just few centimeters apart
- Position 2: distance $\approx 2m$
- Position 3: distance $\approx 6m$

As the UE was moved further away, the connection became way more unstable until it was completely lost. This instability is a common issue throughout the whole thesis work.

Since the UE and the gNB couldn't ping each other, it was therefore impossible to measure latency, and most importantly, it was not possible to establish a connection that allowed the deployment of a real application across the 5G network. For this reason, OAI MEP was discarded and the 5G network was restored to its original configuration described in Chapter 2.

3.4 Final considerations on OAI MEP

As the experiment showed, OAI MEP could be a powerful tool to get RAN KPIs while also deploying an application.

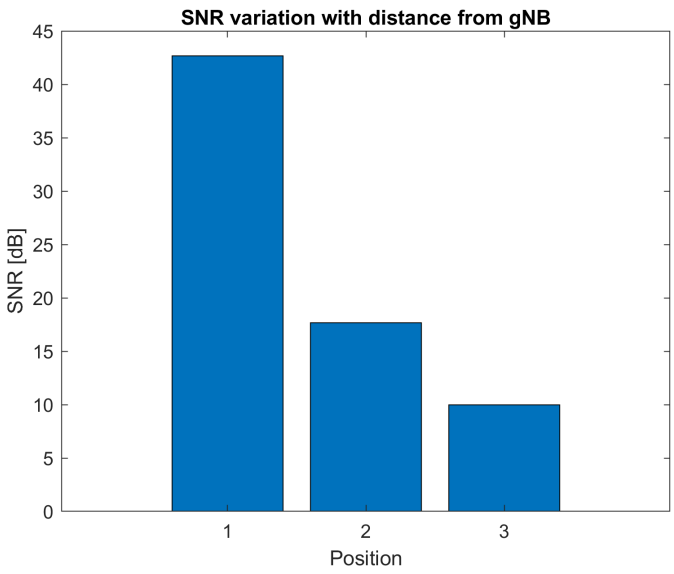


Figure 3.5: Average SNR at different distances

Unfortunately, as of today, this tool is not ready yet to be used in an actual application, meaning that other solutions must be found to retreive the KPIs until OAI menages to complete the MEP development.

Chapter 4

ROS 2 framework

4.1 ROS 2 overview

The Robot Operating System (ROS) is a collection of software libraries and tools designed for developing robot applications. It provides a broad set of commands and customizable options to control a wide range of robots, from stationary to mobile robots, in a multitude of cases.
ROS is released in distributions, commonly referred to as "distros", with multiple distributions supported simultaneously. Some releases offer long-term support (LTS), ensuring greater stability and extensive testing. Other distributions are newer with shorter lifespans but provide support for more recent platforms and up-to-date versions of their constituent ROS packages.
ROS 2 [6] is the second version of this operating system and was launched in late 2017.

The main concepts are:

- Nodes: an entity that uses ROS to communicate with other nodes.
- Messages: a ROS data type used for subscribing or publishing to a topic.
- Topics: where nodes can publish messages or subscribe to receive messages.
- Discovery: the automatic process that enables nodes to find and communicate with each other.

A scheme of the working principle is reported in figure 4.1.
A node is responsible for one aspect of the robot, like managing the wheel or controlling a sensor. Each node can send and receive data from other nodes via topics, services, actions, or parameters.

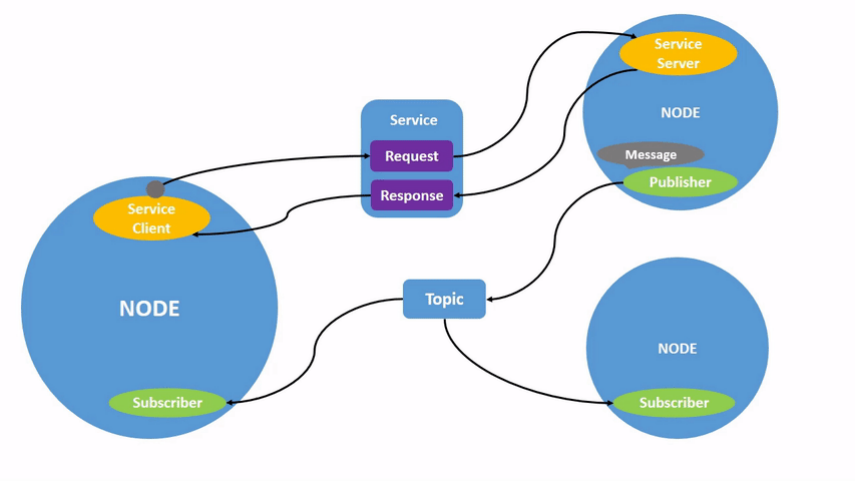


Figure 4.1: ROS2 working scheme [7]

Nodes publish information over topics, which allows any number of other nodes to subscribe to and access that information. Topics, therefore, are not just point-to-point exchanges.
Services are another form of communication between nodes; they provide data only upon request by a client and not in a continuous way. There can be many clients but just one server per service.
Actions are an alternative communication type and are intended for long-running task. Their working principle is similar to the service's one, with the difference that actions also provide a feedback.
Parameters are configuration values of a node. They are particularly useful because allow to save a configuration and reload it when needed.

4.2 ROS 2 application development

For the thesis work, the *Humble Hawksbill* distro was employed, coupled with Ubuntu 22.04. To test the effects of the 5G connection on the robot, it was necessary first to develop a ROS 2 application that could be affected by such effects. It was decided to create a program that made the robot follow a specified path exploiting a feedback action.
This choice was made because an application such as this one can convert latency into trajectory error. Moreover, it was ideally simple enough to track the movements of the robot to check for errors. In fact, in case of high latency, the control over the robot would become less effective as the commands would arrive late, causing the path described by the robot to differ from the ideal trajectory.

4.2.1 Gazebo simulator environment

Gazebo was employed to simulate the robot, which is a simulation environment that supports ROS integration, enabling the automatic conversion of messages between the two environments via bridges.
Gazebo provides many different robot models and predefined worlds to the user, as well as the possibility to create a personalized 3D robot model that matches an existing robot. The simulator offers a realistic physics engine able to simulate physical interactions with a high degree of fidelity.
An additional feature that grants the personalization of the robot applications is the possibility to include plugins: these are custom additions that can be used to modify the parameters of the robot and the world, or add some specific feature to the simulator.
Moreover, the integration with ROS and ROS 2 allows the use of the ROS 2 commands to retrieve the simulation's parameters (e.g., publication frequency, topics in use) and to give input to the robot. In Figure 4.2 can be seen how the world file is launched; this file includes the settings for both the world and the robot, plus the initialization of the bridges.

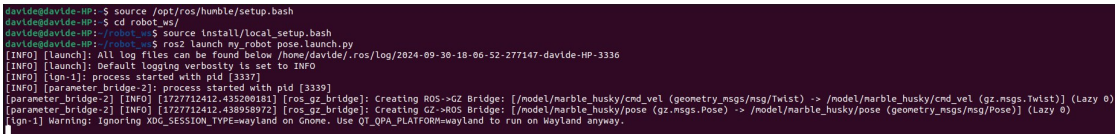


Figure 4.2: Launch commands

Figure 4.3 shows the sequence of input commands used to provide feedback action to the robot, as well as the automatic velocities commands provided to the robot by the script.

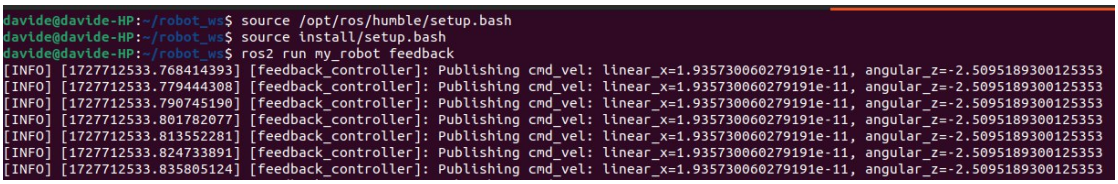


Figure 4.3: Robot's movement commands

4.2.2 Robot model selection and path definition

The initial step for the development of the robot application consisted in selecting a suitable robot model, and an Ackermann steering robot was chosen for its steering capabilities and higher achievable speed in comparison to other dynamic models such as a differential drive. The first model of choice was the basic Ackermann steering plugin model present in Gazebo.
After the initial choice of the model, it was necessary to establish a connection between ROS 2 and Gazebo, i.e, a bridge. This procedure can be done by creating a bridge for the required ROS 2 topics in the simulation launch file.
The main issue in executing this procedure was the presence of three sets of Gazebo commands, each corresponding to a different version of the simulator. Gazebo, in fact, released a newer version of the simulator called Ignition, then rebranded it Gazebo and changed the majority of the commands, making it confusing to find the required commands between the three versions (Gazebo Classic, Ignition, Gazebo). Finally, it was possible to successfully create the two bridges, one for speed commands and the other used to retrieve the pose of the robot.

Then, it was necessary to define the path to be followed by the robot that better fits the project goal. The choice ended up being an 8-like path.
This trajectory was chosen for its relative simplicity and because it respected the requirements for the 5G experiments (i.e., need for low latency). A circular shape, despite being simpler to implement, needed just one speed command composed of a linear and an angular speed to follow the path; this meant that latency wouldn't affect the robot's movement once the command was given to the robot, as even an extremely high latency wouldn't cause the robot to deviate.
The first option to define the 8-like path consisted of creating it by making two circles intersecting at just one point. To make it look like an 8, the angular speed was inverted when the robot reached the center of the desired figure.
This way of producing the path was discarded because there was just one point that could be affected by errors and was not possible to achieve good trajectory tracking. The reference to define the path was founded in [8], and is defined as:

$$\begin{aligned} x &= A \cdot sin(t) \\ y &= B \cdot sin\left(\frac{t}{2}\right) \end{aligned} \tag{4.1}$$

Once the reference path was defined, it was necessary to retrieve the pose of the robot to implement a feedback action. This procedure was not straightforward as the simulator published pose information about all the single components of the robot (e.g., wheels, joints), making it difficult to process the data. Seven measurements (three for the position and four for the orientation) were published

every 0,001 seconds for all the components of the robot, filling the log with data that was hard to interpret.
Different types of messages were tried to retrieve the required data, but none of them satisfied the requirements.
To successfully get the pose, I decided to modify the internal plugins of the robot and the simulated world. To do so, it was necessary to get a personalized robot model and create a world file.
The model of choice was taken from the list of models available on the Gazebo website; it [9] is an Ackermann steering robot characterized by medium-high agility and speed.
The model was placed in the world file along with the ground and the light. Additionally, a plugin (refer to Appendix A.2) was included to address the pose issue. This plugin enabled the publication of only the pose of the robot's chassis, which serves as the reference point. The pose comprising a (x, y, z) Cartesian position and a quaternion orientation representation is sent to the ROS 2 program. An illustration of such can be seen in figure 4.4.

```
name: "marble_husky"
position {
  x: 0.0051718365966708308
  y: -0.001208210139709516
  z: 0.074999993149118691
}
orientation {
  x: -4.9792099589066079e-10
  y: -9.4755442216174861e-11
  z: -0.0028420896053718584
  w: 0.99999596125518175
}
```

Figure 4.4: Example of the pose representation

This file also included some parameters regarding the pose publication frequency that will be treated in the next chapter, as their influence is particularly significant when the application is deployed across the 5G network.

4.3 ROS 2 application deployment

The Python program I created to control the robot generates waypoints for the 8-like trajectory and subscribes to the robot's pose while publishing velocity commands to follow the waypoints. The control loop computes the required linear and angular velocities based on the difference between the current pose and target waypoint, applying proportional control gains. The node logs the robot's pose

and adjusts its velocity in real-time to minimize the distance and angular errors, ensuring the robot follows the figure-eight path accurately. The script includes helper functions for quaternion to yaw conversion and angle normalization, and it handles proper node shutdown on interruption.
Figure 4.5 compares the desired ideal curve and the actual one when the commands are given at 10 Hz. In this case, the parameters A and B were set to 1.5 and 3, respectively.

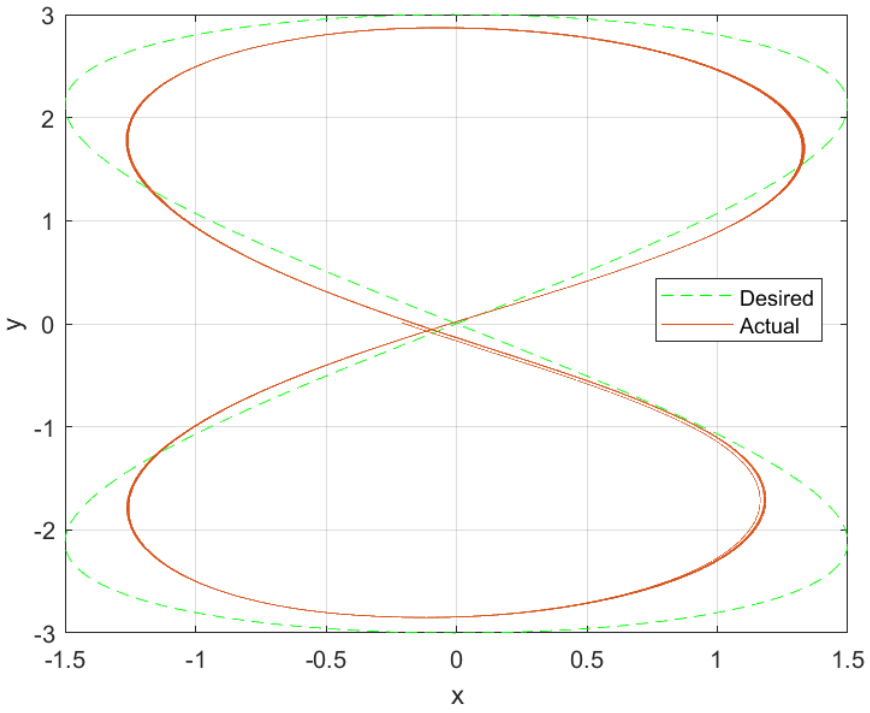


Figure 4.5: Ideal trajectory (green) and actual trajectory (red)

This model could follow an 8-like path with a tracking error of 3 decimeters at maximum, which was considered to be acceptable also because of the robot's ability to repeat the same figure with negligible differences between one cycle to the other. The robot's kinematic constraints (i.e., how much can the wheels turn), however, made it impossible to achieve a perfect following of the path, especially at the edges of the x-direction in correspondence with the maximum radius of curvature. This issue can be considered negligible for this thesis as the aim was not to design an accurate control law.
Different tests were performed to find the best values for the gain and amplitude values. Other experiments were conducted to study the effects of the command publishing frequency on the capability to follow the path, and the results are

presented in the next sections.

4.3.1 Amplitude experiments

To study the effects of the parameters A and B in equation 4.1 on the capability of the robot to follow the path, different tests with publishing frequency of the commands at 10Hz were performed. A and B change the sizes of the 8, along the X-axis and the Y-axis respectively.
At first, A and B were set to 1, but these values caused the desired path to be too small, making it impossible for the robot to follow such a tight trajectory.
Successive experiments were carried out to find an optimal value for A and B. The first attempt was made with A = 2 and B = 3; in this case the robot could follow the trajectory, but the time required to complete the figure with good enough results was over 30 seconds. Other experiments were performed with values of A and B going from 1 to 3.
The best result in terms of time and trajectory tracking was achieved with A = 1.5 and B = 3.

A second series of trials were made to tune the proportional gains. The best results were obtained with $K_{P,linear} = 1, K_{P,angular} = 0.8$.
In figure 4.6 is shown an example where the $K_{P,linear}$ becomes too high, leading to an unstable system.

Figure 4.7 shows the results of these experiments. As clearly visible, there's an initial error caused by the initial pose of the robot; in fact, it spawns at (0,0), so it needs a first half-cycle to get to the correct orientation where the feedback control becomes effective.

4.3.2 Publication frequency experiments

This section will investigate the effects of commands' publication frequency.
Three tests were performed taking into account the results obtained in the previous section in terms of amplitudes and gains:

- Publication frequency = 1 Hz
- Publication frequency = 10 Hz
- Publication frequency = 100 Hz

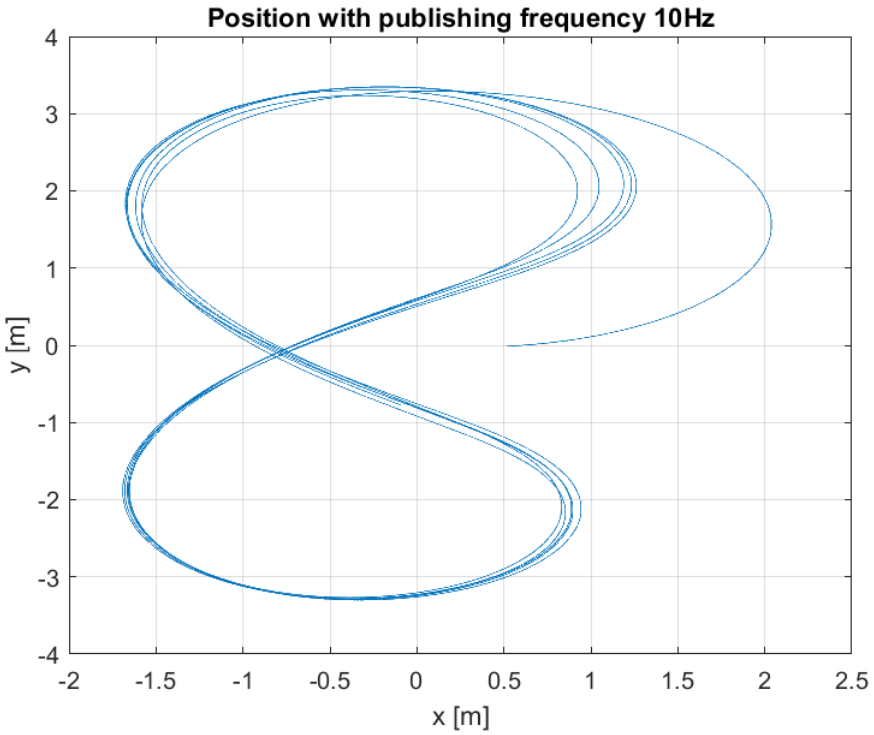


Figure 4.6: Example of an unstable system

It was possible to modify the frequency value by changing a specific parameter in the Python code I created to control the robot. This parameter defined the time between two consecutive pose readings, and every single reading corresponded to a velocities command sending.
When the frequency was set to the lower value, the worst results were obtained. This outcome can be explained by the poorly frequent corrective messages sent by the controller to the robot.
The position of the robot with this frequency value can be seen in figure 4.8.

Publishing frequencies higher than 100 Hz for the velocities commands were not tested as 10 Hz already provided a good result, and moreover because the system could not bear high frequency messages exchange.
Overloading the system could lead to crashes or failures due to limited computational power.
At a publication frequency of 100 Hz, the path followed by the robot, as shown in Figure 4.9, closely matched the path obtained at a frequency of 10 Hz (Figure 4.7), with an error reduction in the order of the millimeters. This minor improvement could be attributed to the relatively low speed of the robot, which implies that fast

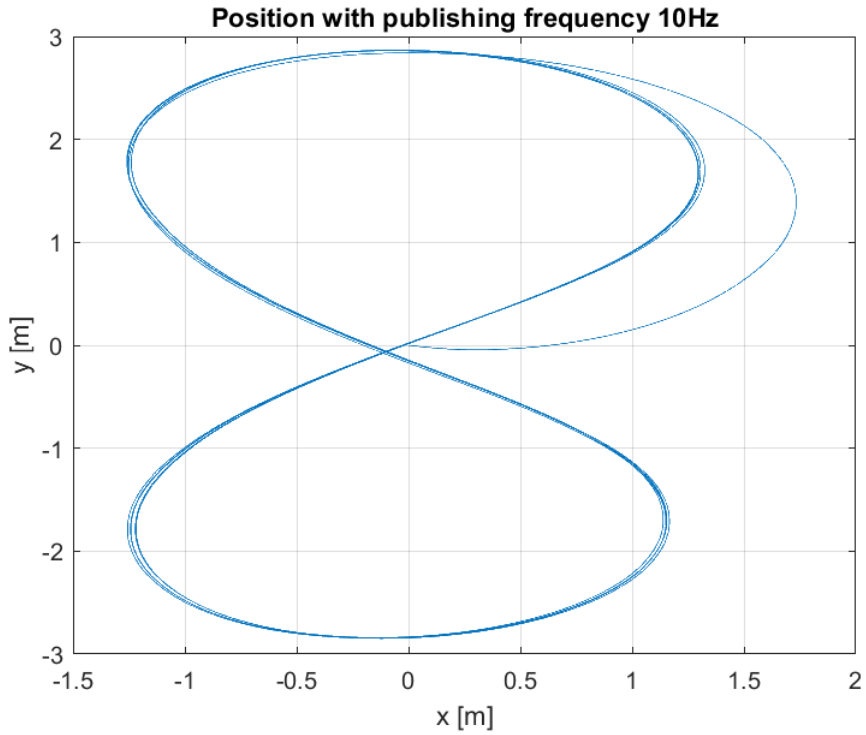


Figure 4.7: Best trajectory achieved

corrective actions are not useful.
Consequently, the 10 Hz frequency was chosen as a reference due to its small tracking error and the reduced workload on the system, resulting in fewer messages being processed.

Figure 4.8: Trajectory with publishing frequency of 1 HZ

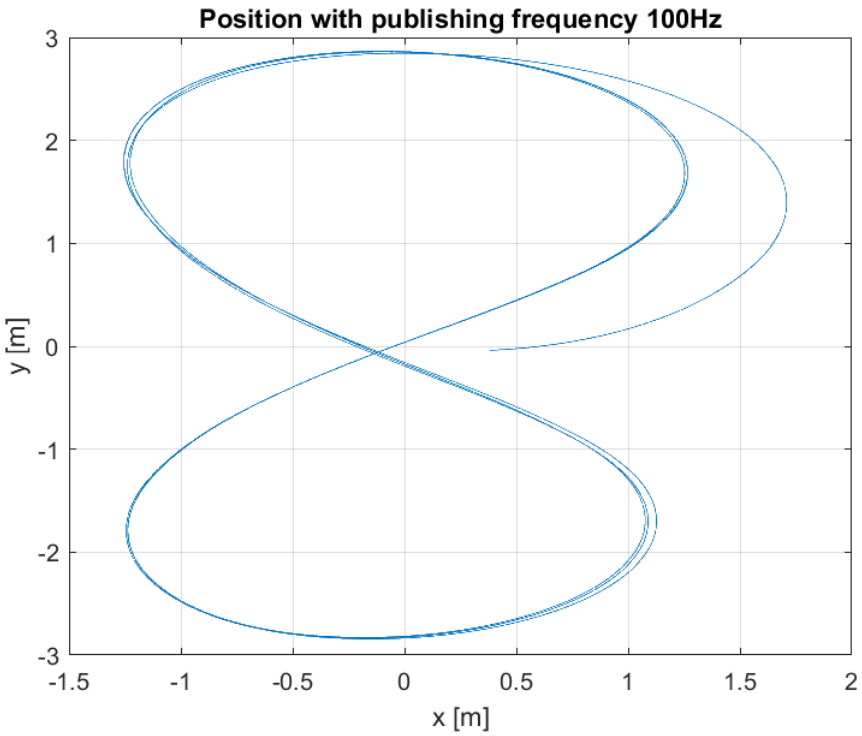


Figure 4.9: Trajectory with publishing frequency of 100 HZ

Chapter 5

Implementation of ROS 2 controller across the 5G network

5.1 Deployment of ROS 2 nodes on 5G

To have a full deployment across the 5G network, it was necessary first to establish a bridge between the ROS 2 messages and the OAI network, and for this purpose, Zenoh was employed.
This tool was used because the OAI 5G network did not support UDP multicast messages, which were employed by the DDS middleware [10] employed by ROS 2 [11]. Moreover, Zenoh is suitable for low-latency, high throughput, such as 5G.
Zenoh allowed the establishment of a TCP connection between the two known devices (i.e., UE and gNB) and the exchange of ROS 2 messages.
An example of how this bridge works is shown in figure 5.1.
In my case, one PC was hosting the gNB and the robot controller program, while the other one included the UE and the Gazebo simulation.

The first procedure required to create this bridge was to redirect the gNB messages towards a specific address reachable by the UE. This was done using the command *ip route*, provided by the command line.
Then, it was possible to create a peer-to-peer connection between the two PCs, allowing the detection of the ROS 2 messages being exchanged by the nodes.
When attempting a full deployment with the Gazebo simulator running, the simulation crashed due to the UE's insufficient performance.
To solve this problem, it was necessary to rethink the setup: a third PC was

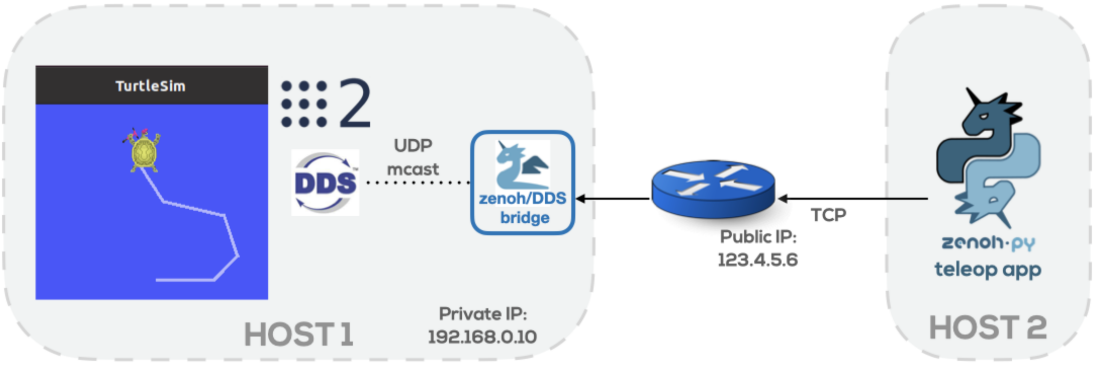


Figure 5.1: Zenoh bridge [12]

required to host the simulator to reduce the payload on the UE. To route the ROS 2 messages from the UE to the simulator, an Ethernet-cabled connection was used. Successful message exchanges between the PCs connected via Ethernet were ensured by ticking off an option in the connection settings menu; in particular, enabling the *share with others* setting in the IPv4 menu.

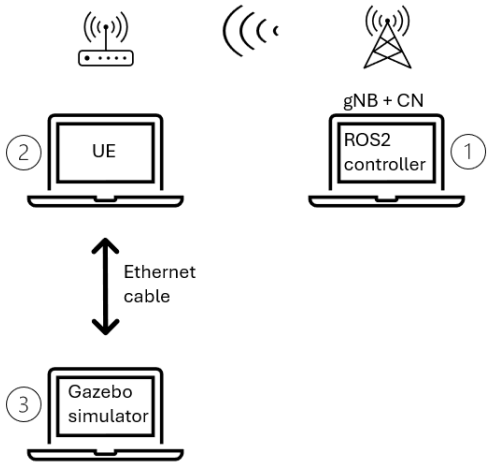


Figure 5.2: Three PCs setup

With the setup shown in Figure 5.2, it was finally possible to test the effects of latency on the robot control.
To deploy the ROS 2 nodes the following steps were required:

1. Connect UE to gNB following the same procedures used in Chapter 2
2. Start the Zenoh bridge on both the computers 1 and 2

3. Launch the ROS 2 world on PC number 3

4. Run the robot controller on PC number 1

After completing all these steps successfully, latency, pose, and bandwidth measurements could be performed.

5.2 Performance measurements

The first verification that was performed consisted in comparing the robot's position when the control is deployed across the 5G network against the simulated behavior obtained with the whole ROS 2 system on one PC.
This test was performed with pose publication frequency set to 1 kHz, while the velocities commands were published at 10 Hz. The results are shown in Figure 5.3. As clearly visible, the simulated behavior is more stable and repeatable; whereas the actual one is characterized by a larger variance at every cycle.
In the next phases of this thesis job, the effects of the publication frequency on the robot performance will be studied, plus other tests that aim to study the behavior of the system as some parameters (e.g., distance between UE and gNB, latency test setup) change.

5.2.1 Pose publication frequency

Since the pose publication frequency of the robot was set to 1 kHz, and since this value was very high with respect to the commands' frequency, this parameter was the first to be investigated.
This value was set as default in the world file I created to spawn the robot, and it was possible to change it by modifying the `<max_step_size>0.001</ max_step_size>` and `<real_time_update_rate>1000</ real_time_update_rate>` parameters in the world setting (see Appendix A.3).
The comparison was made between the pose publishing frequency at 1 kHz, from now on referred to as Standard, and a reduced frequency equal to 100 Hz, called Reduced. Exploiting the command *ros2 topic hz* that automatically provided the actual publication frequency of a certain topic, it was possible to see that the Standard frequency was published approximately at 880 Hz. The Reduced one was instead correctly measured at 99.9 Hz.
This error in the Standard frequency is due to the excessively high publication frequency, which combined with the lack of computational power of the UE, made it impossible to process data at such high speed.
The results of the two tests can be seen in Figure5.4.
The trajectory error visible on the right-top part of the Reduced frequency path is

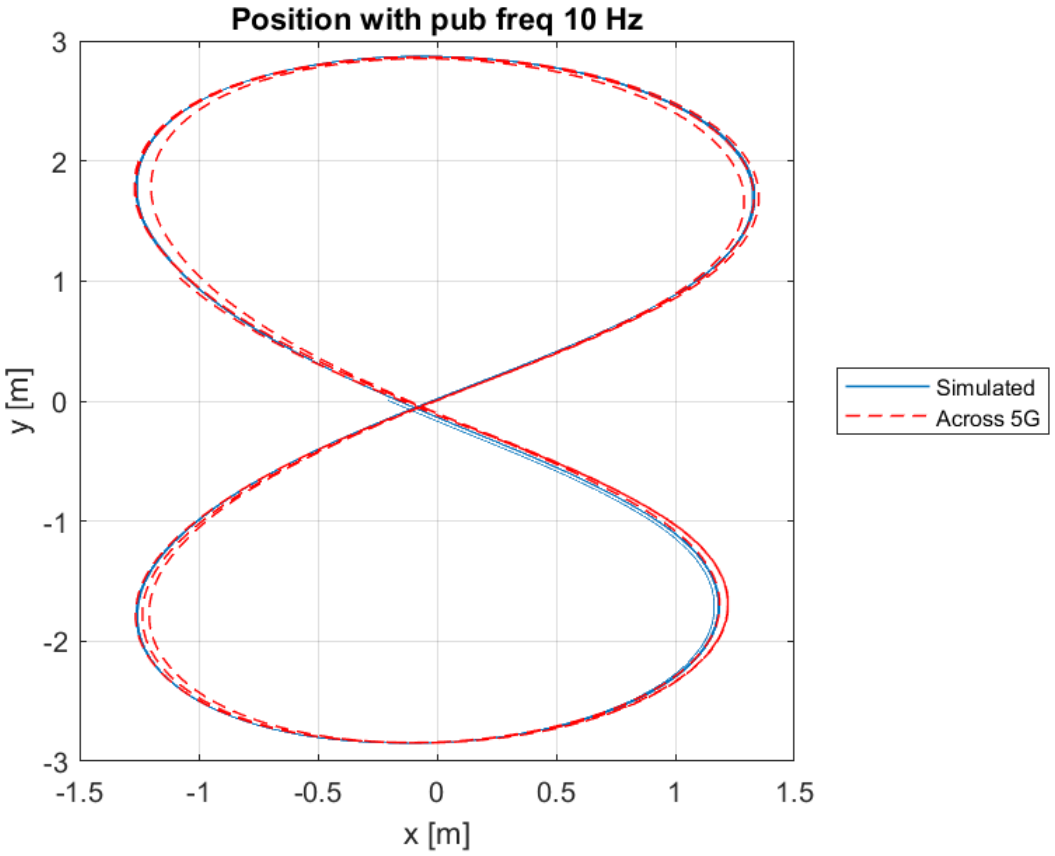


Figure 5.3: Simulated position vs actual position

due to a loss of connection, meaning that it does not appear in normal conditions. It was left in the figure to show what happens when the connection is lost: the robot doesn't receive any updated command and, therefore, keeps executing the last received command eventually performing a circle-shaped path.
The trajectory errors related to the two paths has been calculated comparing them to the reference path already shown in Figure 4.5.
The results of these computations are reported in Figure 5.5
The nearly period and oscillating behavior of the trajectory errors are to be attributed to the particular 8-like path followed by the robot, which causes smaller errors when the robot is at the center and larger errors at the sides while decreasing at the top and bottom. One full 8-shaped cycle is completed in 30 seconds, meaning that the lower peaks for each error will occur every 15 seconds (i.e., every time the robot passes the closest to the origin).
Figure 5.5 illustrates that the errors along the Y-axis are nearly identical for both

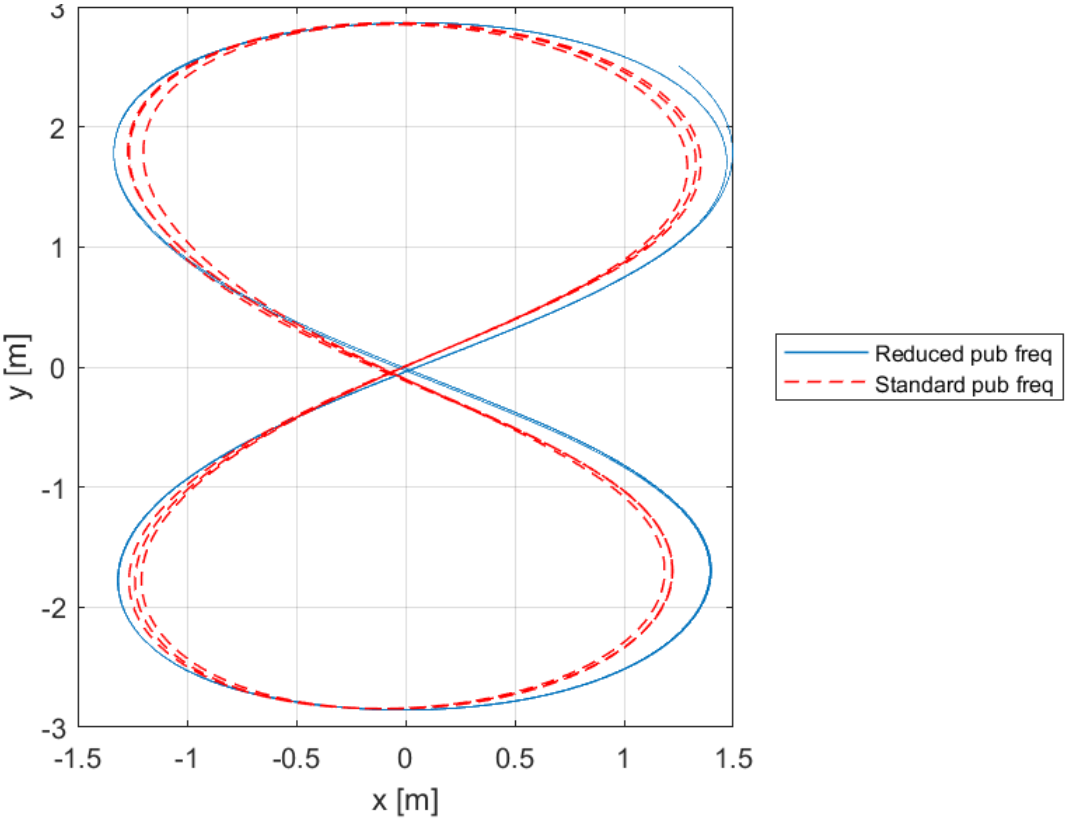


Figure 5.4: Robot's position with pose publishing frequency 1kHz and 100Hz

paths, differing by only a few centimeters. However, the differences along the X-axis are more significant, reaching several decimeters.
This initial analysis clearly indicates that a reduced publishing frequency enhances trajectory tracking performance. To further understand and to justify the differences between the two cases, other experiments were carried out.
The first set of experiments regarded the measurement of the latency. This operation was performed by writing a pair of C++ scripts, called Talker and Listener. The Talker file defines a ROS 2 node that publishes messages on a topic, listens for responses on another topic, and logs the round-trip latency (the time between sending a message and receiving the response) into a file. Specifically, it saves the timestamp at which the message is published and compares it with the timestamp of the received response to calculate the RTT.
For the first test, it was decided to measure latency (i.e., to send a message) once every second.
The Listener, instead, sends a response every time it receives a message from the Talker. It is possible to have more than one Listener differentiated by a specific

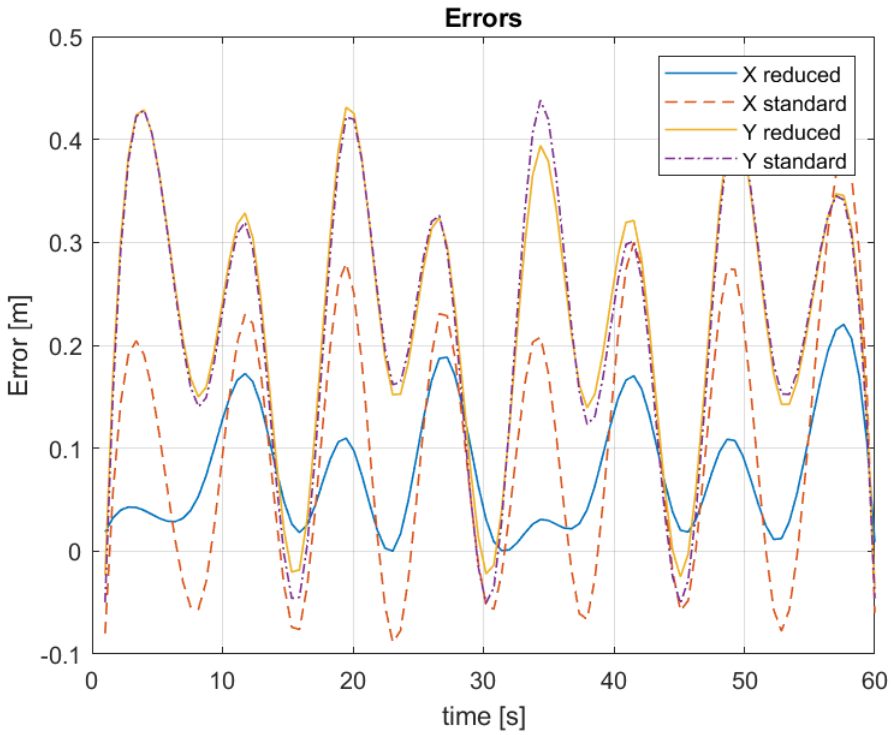


Figure 5.5: Trajectory errors along X and Y

name.
For these tests two listeners were needed, one hosted on the PC with the Gazebo simulator, and one on the UE.

The first test aimed to verify the latency introduced by the Ethernet connection between the UE and the simulator, and to check if it could influence the robot's operation. As expected, the latency introduced by the cabled connection was negligible.
This is confirmed by Figure 5.6, which shows the latency values from the gNB to the UE and from the gNB to the simulator; the maximum difference between the two is about 1 ms.
These measurements were performed with the simulator running and with a pose publishing frequency of 1kHz.
A second test was carried out to investigate how the simulator could influence the RTT. Two latency measurements from gNB to the simulator were performed, one with Gazebo running and one without the simulator executing any operation. Both these experiments were performed with a Standard publication frequency.
As visible in Figure 5.7, the RTT is higher when the simulator is running, with an

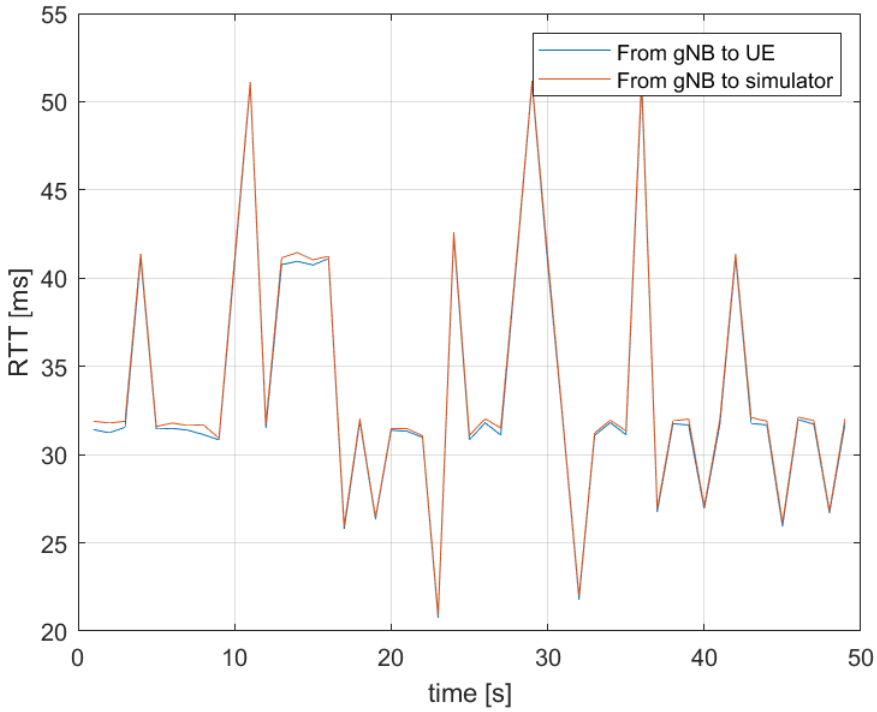


Figure 5.6: RTT measurement

average value (40 ms) about four times higher than the idle case.
The same test with Reduced publishing frequency provided an average RTT with the simulator running of about 22 ms, meaning a halving in the latency.
The spiking phenomena visible when the simulator is running will be investigated later.

From this first analysis appears evident that the Reduced frequency provides better trajectory tracking and reduced latency, meaning an improvement in the two most important performance parameters.
The fact that a smaller publication frequency reduces the latency can be explained by the reduced workload induced by the system on the UE, which is the PC most affected by performance issues. In fact, limited computational power could lead to problems while processing packets, causing retransmissions and errors.
Therefore, less frequent packets can be processed correctly leading to smaller RTT and a better control over the robot movement.
To verify the throughputs in different frequency conditions, some other experiments were performed exploiting a ROS 2 command. The *ros2 topic bw* command provided the amount of data trasmitted every second on a certain topic.

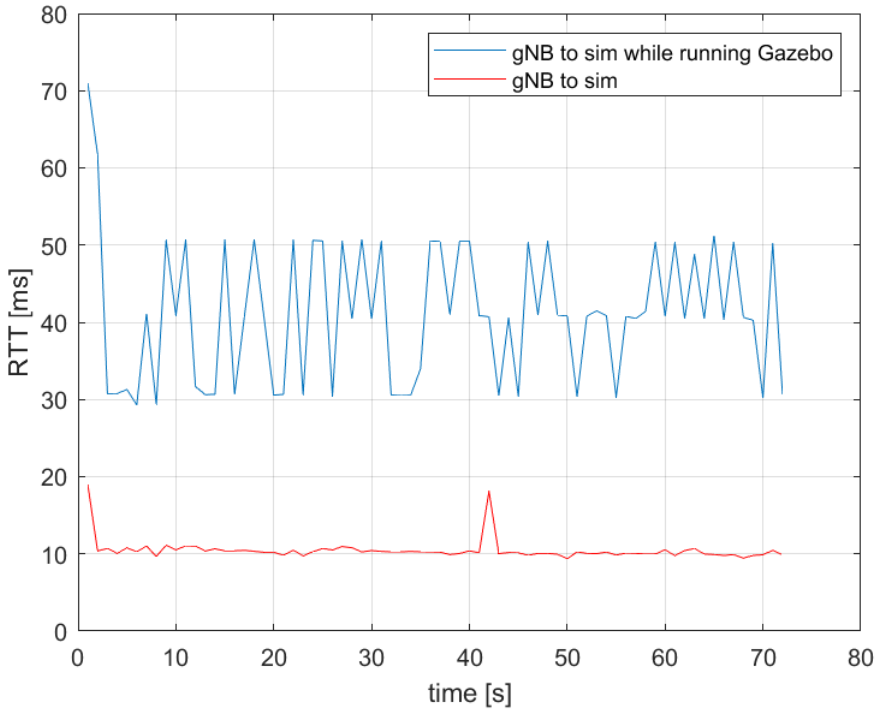


Figure 5.7: Latency

As Table 5.1 shows, reducing by a factor of 10 the pose publication frequency reduces the pose BW of 1/10 as well. This drop in the BW can explain the better performances of the Reduced frequency case: less congestion across the network and less workload demanded to the UE, leading to faster execution of the velocities commands and so, better trajectory tracking.
It can also be noticed that in the case of Reduced frequency the velocities commands' bandwidth slightly increases. This is most likely due to the reduction of the pose BW, which allows for a better and denser transmission of velocities commands' packets.

	Standard frequency	Reduced frequency
BW pose [KB/s]	54.54	5.99
BW cmds [B/s]	512.32	518.23

Table 5.1: Bandwidths at different publishing frequencies

5.2.2 Velocities commands publication frequency

The second set of experiments concerned the commands' publishing frequency. Two frequency values were taken into account: 10 Hz and 100 Hz.
It was possible to switch between these two quantities in the same way as explained in section 4.3.2 (i.e., by changing a specific parameter value in the script).
Values below 10 Hz and above 100 Hz were discarded because of the results obtained in the simulated environment on just on PC.
Using the *ros2 topic hz* command, it was seen that the actual publication frequency of the commands was 9.99 Hz in the first case, and 94 Hz in the second one.
Figure 5.8 shows the different paths followed by the robot when using the two commands' publication frequencies in the Reduced pose frequency case. As visible, the difference between the two paths is small, with a maximum separation of 5 centimeters in the bottom-right part and an average difference in the order of the millimeters in the rest of the figure.
The 10 Hz option performs slightly better than the 100 Hz one. This difference can be explained again by the lower traffic induced by the smaller frequency, which still provides good control over the robot's movement.

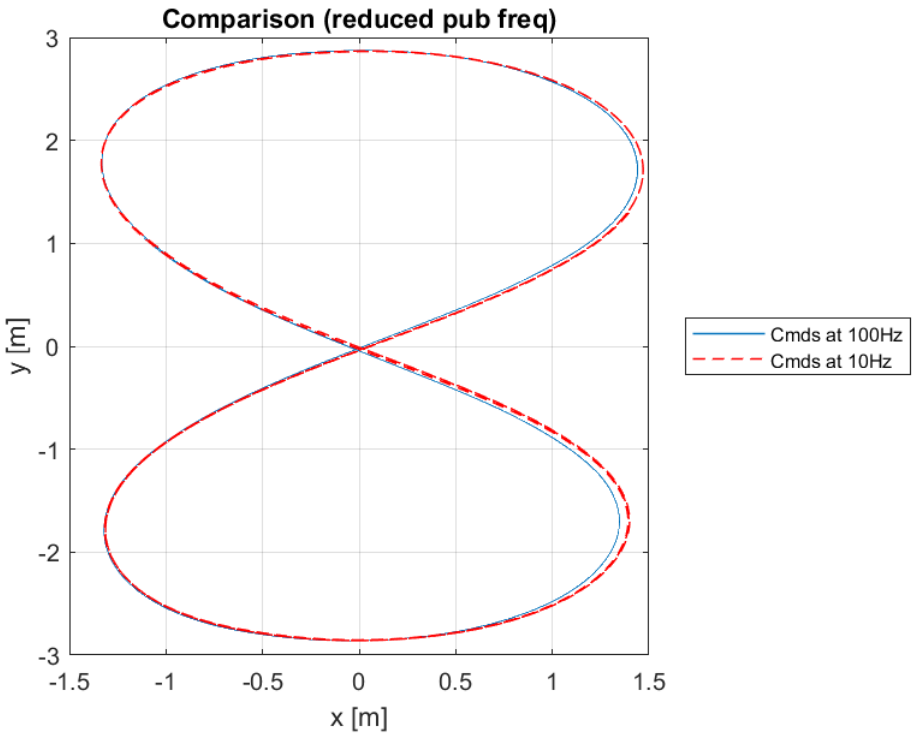


Figure 5.8: Position at different commands publishing frequencies, Reduced case

In Figure 5.9 the robot's paths are reported in the Standard case. In this case, the separation between the two curves is significant, reaching several centimeters on average.
Also in this experiment the 100 Hz case performed worse, causing different crashes of the 5G network before having a stable enough connection that allowed for the measurements to be performed.
Indeed, the combination of elevated pose and commands publishing frequencies is almost unbearable for the system, which tends to crash. Moreover, it becomes difficult to execute other measurements simultaneously as it means adding further workload to an already unstable system.

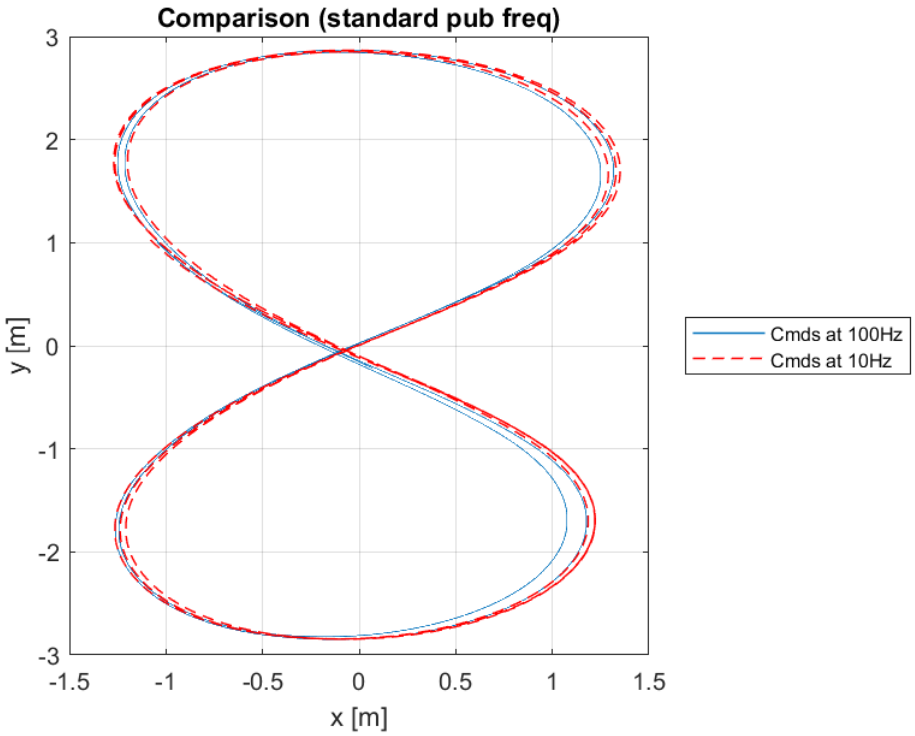


Figure 5.9: Position at different commands publishing frequencies, Standard case

As for the RTT, the results are:

1. Reduced pose frequency:
   - 10 Hz commands: RTT = 22 ms
   - 100 Hz commands: RTT = 28 ms
2. Standard pose frequency:

- 10 Hz commands: RTT = 40 ms
- 100 Hz commands: RTT = 51 ms

The increase in latency when using the higher commands publishing frequency is due to the increased amount of data that the UE has to process, which leads to greater RTT.
Exploiting the *ros2 topic bw* command, it was possible to measure the different throughputs, as reported in Table 5.2. The pose's BW is not reported as it is not

	Standard frequency	Reduced frequency
10 Hz cmds	512.32	518.23
100 Hz cmds	4203.75	4267.30

Table 5.2: Commands' bandwidths at different publishing frequencies

influenced by the commands' publishing frequency, meaning the values are the same ones reported in Table 5.1.
In both 10 Hz and 100 Hz cases, the Reduced frequency produces a higher throughput with respect to the Standard. This means that at equal commands' frequency, the Reduced case performs better control over the robot, as more frequent feedback commands are sent.
Considering the Reduced frequency, even though the 100 Hz case produces more frequent messages, they are counterbalanced by higher latency that makes the overall performance slightly worse. Moreover, since the robot moves quite slowly (maximum speed is below 5 m/s) highly frequent commands may be unnecessary.

5.2.3 Latency test implementations

To study how different ways of measuring latency could affect the performance of the system, two Talker implementations were compared.
The first was the one used in all the previous experiments, meaning that the latency test was in a separate script with respect to the ROS 2 robot controller. In the second implementation, both the latency test and the controller were included in a single file.
To compare the two tests, RTT, BW, and the robot's position were recorded.
These tests were performed in the Reduced pose frequency case and with velocities commands at 10 Hz, as these were the best possible conditions as seen in the previous sections. The two implementations were compared with the no-latency case and then with each other.
From now on the three different possible setups will be indicated as:

1. No latency test performed
2. Latency test performed in a separated file with respect to the ROS 2 controller
3. Latency test and controller on the same script

When comparing cases 1 and 2 (Figure 5.10), it is evident that the latency test performed independently from the ROS 2 controller has no significant impact on the robot's position. The average distance between the two curves, in fact, is below 1 millimeter. Also the commands' bandwidth shows no meaningful difference, with an average value of 519 B/s.
The fact that the BW difference is close to none is due to the small amount of data transmitted by the Talker and the Listener, which leads to a small overhead when processing the packets. The throughputs of these two ROS 2 nodes have been measured and the results are:

- Talker BW: 57 B/s
- Listener BW: 64.96 B/s

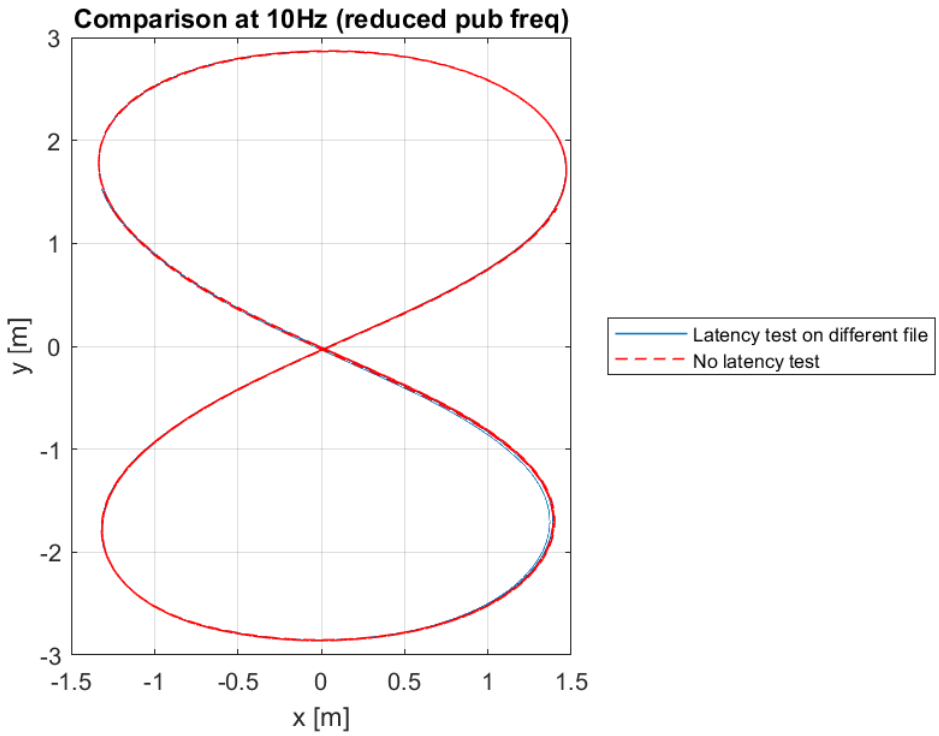


Figure 5.10: Robot's positions in case 1 and 2

The comparison between cases 1 and 3 shows a different situation, with evident separation between the curves in the whole right part of Figure 5.11. Also the errors of the two paths show significant differences, as visible in Figure 5.12.
The X-axis error related to case 3 is higher than the one of case 1; in particular, it is higher in the points corresponding to the center of the 8-like shape. As for the Y-axis errors, the case 3 error appears to be greater in almost every point except the ones corresponding to the top-right extreme (i.e., the third peak of every cycle), where the case 1 error reaches higher values.
The differences in both X and Y axis errors are not negligible, making case 3 the worst-performing setup of the experiment.

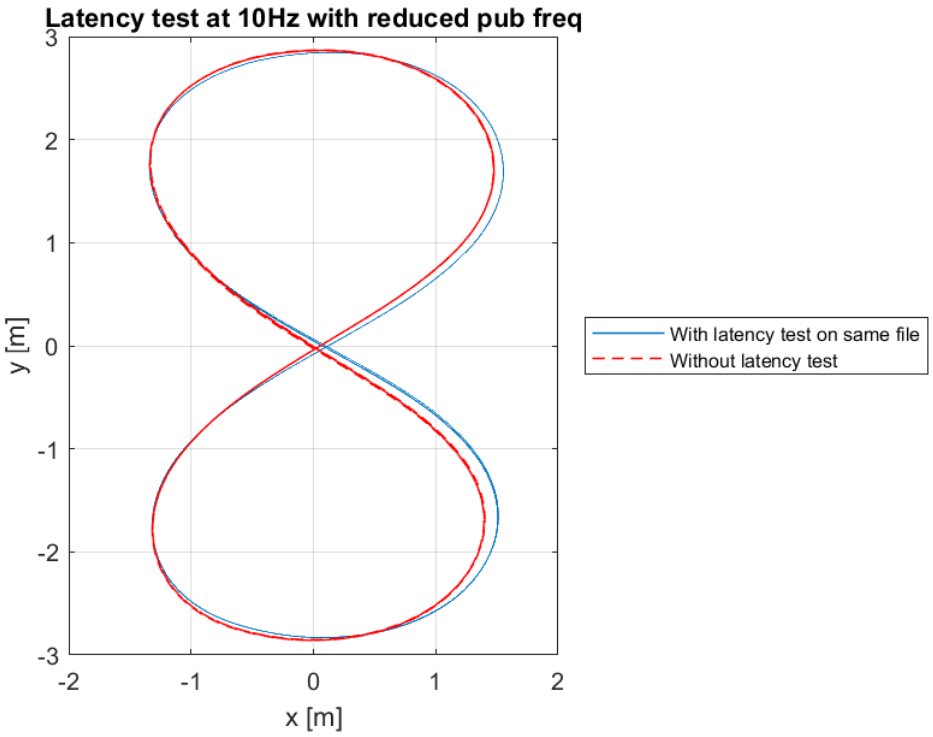


Figure 5.11: Comparison between case 1 and 3

The reasons behind the poor performance of the case 3 setup are explained by analyzing the RTT and the BW.
The latency measurements obtained with this configuration provide an average RTT of 102 ms and an actual publication frequency of the commands' messages of just 4 Hz (value obtained using *ros2 topic hz*).
To further understand this lack of performance, the latency test was carried out also

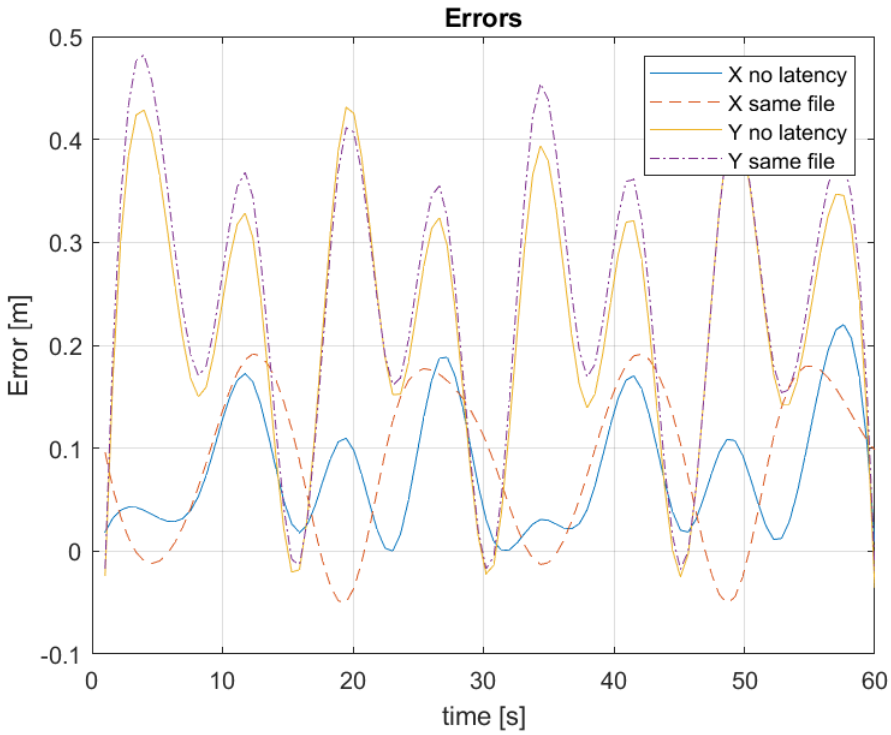


Figure 5.12: Robot's position trajectory errors in case 1 and 3

with a publication frequency of 100 Hz, and also in the Standard pose frequency case. All these tests have produced as a result an average RTT of around 102 ms, with just small decimal differences.
This result induces to think that the reason behind the high latency is the ROS 2 script which takes too much time to execute the latency test before sending the speed commands, causing a decrease in the publication frequency.
This reasoning is supported by the BW measurements that show an average bandwidth of 211 B/s, which is less than half as before. This small throughput value translates into few commands exchanged, hence reduced feedback action.

The final comparison regarded the two latency configurations. Considering the results obtained in the two previous experiments, case 2 is the best way to measure latency while running the Gazebo simulator.
Figure 5.13 shows the two paths.

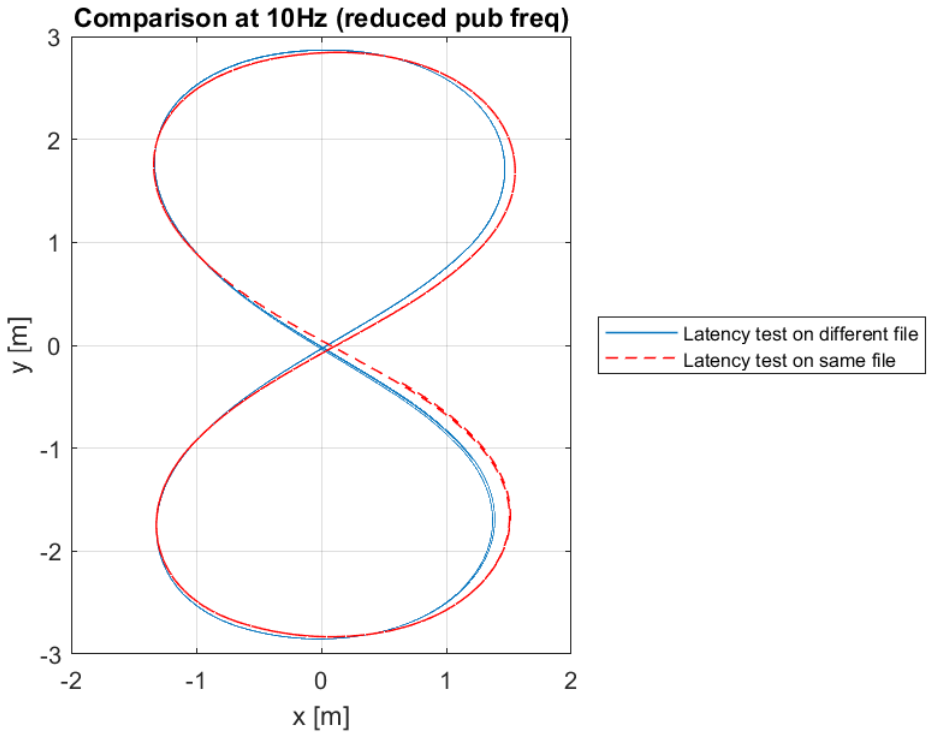


Figure 5.13: Robot's position in the two latency test implementations

5.3 Spiking phenomena analysis

To investigate the spiking phenomena visible in the RTT figures (5.16 and 5.18), it was necessary to get more network performance metrics to get a relationship between RTT spikes and other parameters such as BLER, retransmitted packets, SNR, and so on.
The standard OAI log did not provide such parameters, meaning that other options had to be explored.
The first attempt consisted in modifying the data published by the OAI network, this was successfully done by changing a setting that allowed the publication of all the physical layer metrics. The drawback consisted of the amount of data being published, reaching several Mb every second, thus making it impossible to open and process the data contained in the log file.
The second way consisted in exploiting OAI RNIS, but since this application is based on the same structure as OAI MEP it was not possible to make it work properly, as explained in Chapter 3.
The last approach was based on FlexRIC [13], a software framework that operates as a near-real-time RAN Intelligent Controller (near-RT RIC) and allows the hosting

of external applications called xApps. FlexRIC is compliant with the O-RAN standards.
O-RAN Alliance [14] is a community of several mobile network operators, vendors, and research and academic institutions active in the Radio Access Network (RAN) industry. It is divided into 11 technical Work Groups (WG), and FlexRIC follows the standards set by WG3 [15], which is the group focused on defining an architecture centered around the Near-RT RIC, which enables near-real-time control and optimization of RAN elements and resources through precise data collection and actions over the E2 interface. This interface connects Near-RT RIC to E2 nodes (such as the gNB), and expose RIC Services from one or more RAN components.

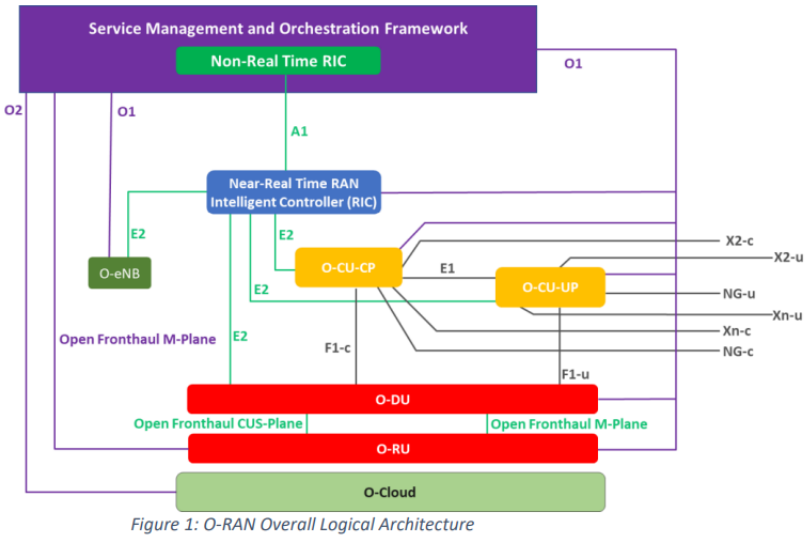


Figure 5.14: O-RAN architecture [16]

To deploy FlexRIC and the E2 Agent, it was followed the OAI tutorial at [17].
For this experiment, the E2 Agent was exploited to deploy an xApp that could measure the channel parameters.
Once the setup phase was successfully carried out correctly, it was possible to measure the KPIs over 5 minutes while also measuring latency through the Talker-Listener pair. These measurements were performed both while the simulator was running and while it was shut down.
Since the measurement of such parameters was computationally demanding, it was not possible to perform high frequency measurements. As a consequence, the xApp was set to perform a series of measurements every 0.001 seconds, and the latency test measured 10 RTTs per second.
From the whole set of measurements, the most relevant were isolated and analyzed.

The main KPIs analyzed were Uplink and Downlink BLER, PUSCH and PUCCH SNR (the first one is related to data while the second to control informations), the aggregated retransmission PRB (Physical Resource Block), and retransmitted packets and bytes. These data were compared to the RTT to look for any correlation that could explain the spikes in the latency plots.
Other metrics employed were the buffer occupation and the transmission waiting time.

- BLER: proportion of incorrect blocks to the total number of blocks transmitted
- SNR: ratio between the signal power and the noise power
- Retransmitted PRB: number of PRBs that have been retransmitted in a given time period, measured at the physical layer
- Retransmitted packets: number of packets that have to be retransmitted, measured at higher layers
- Buffer occupation: amount of data held in th buffer waiting to be transmitted
- Waiting time: time an SDU waits in the buffer before it begins to be transmitted, measured in microseconds

High PRB retransmission is due to poor channel quality, while retransmitted packets could also be due to network congestion or errors in the transmission protocol. The results of the measurements related to the no-simulator case are shown in Figure 5.15; it is visible that all the metrics present stable values as the only active transmission is the one related to the latency test, which is lightweight and concerns small amounts of data.
The only data that seems to strongly relate to the spiking phenomena in the RTT (Figure 5.16) is SNR, both pucch and pusch (Figures 5.15c and 5.15d).
In particular: the spikes at 60 s correspond to a spike in the RTT at the same time instant, the highest peak reached by the RTT at 147 s is correlated to the peak in the pusch SNR.
The only peak present in both BLERs, instead, causes a peak in the RTT.
This means that the inconsistent channel conditions are a reason for the RTT behavior.

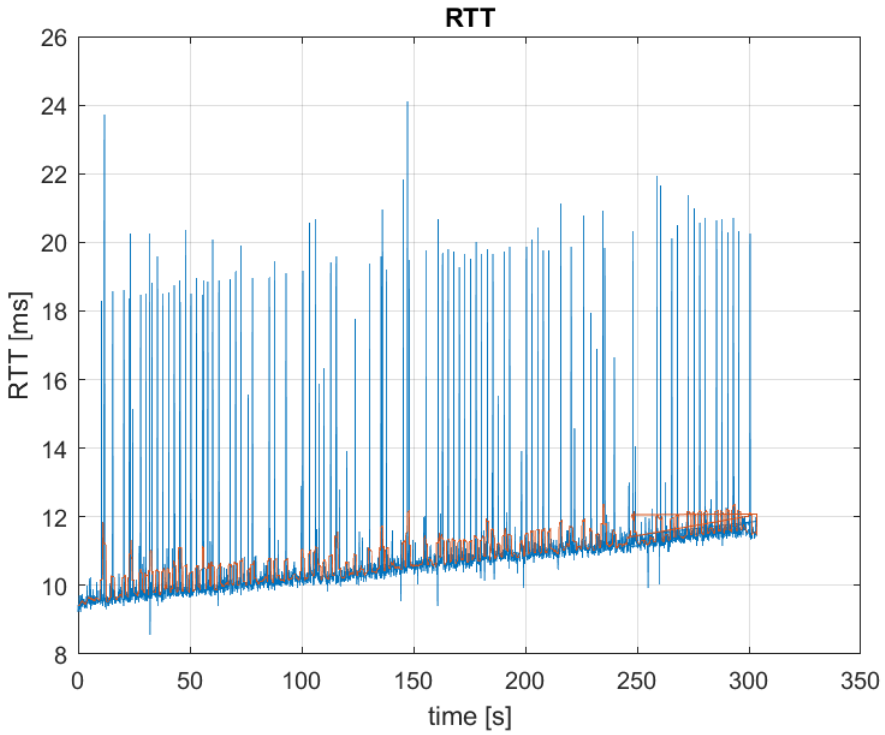


Figure 5.16: RTT in no-simulator case

To better understand the causes of the spiking phenomena, it is necessary to study the measurements related to the robot deployment (Figure 5.17). When the application is deployed, it is clearly noticeable that all the parameters under analysis show a worsening in the conditions. This situation is expected as the amount of data transmitted across the 5G network increases considerably.
From these metrics is possible to detect one particular spike at 250 s that is common to the BLERs, the SNRs and the RTT plot (Figure 5.18). This peak demonstrate that the worsening in the conditions directly affect the latency.
Also considering the other four metrics (e, f, g ,h), is possible to say that the unstable network conditions and the lack of computational power are enough to cause the spiking phenomena. With respect to the previous case, is it also possible to notice that the spikes in the RTT are higher, meaning that poor SNR alone is not sufficient to cause spikes higher than 20 ms.
The increase in BLER can be explained by the high CPU load when running the simulator and the Zenoh bridge.
Another important difference, is the presence of retransmitted bytes and packets (5.17g, 5.17h), that are an additional indicator of network congestion and radio

link quality.
As for the aggregated retransmitted PRBs, is it noticed that are is linearly increasing, meaning that the PRB that are retransmitted are almost constant in time. This also potentially indicates that the network is congested.

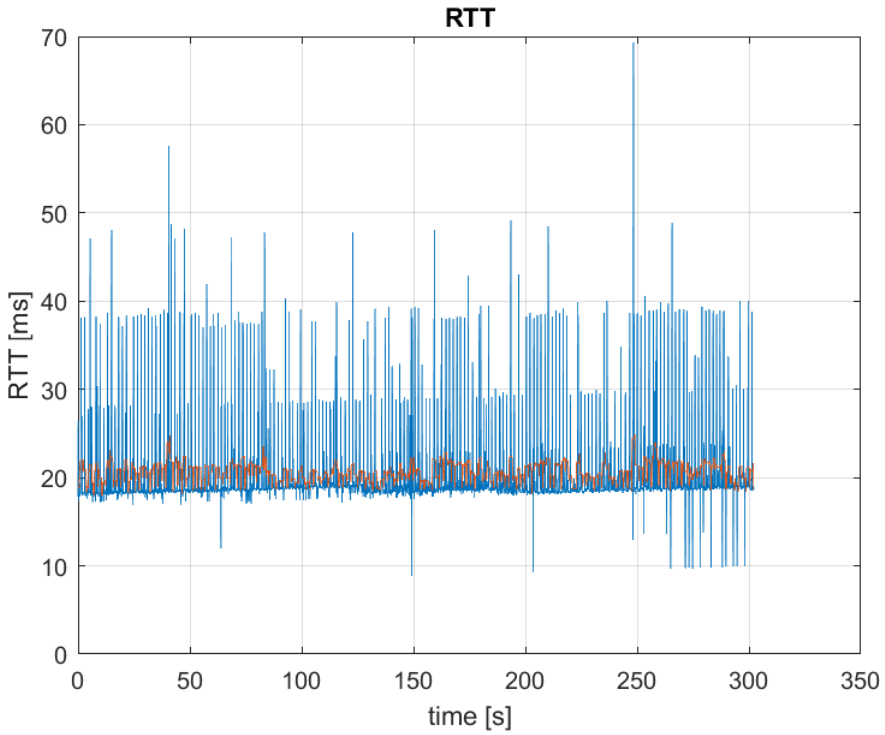
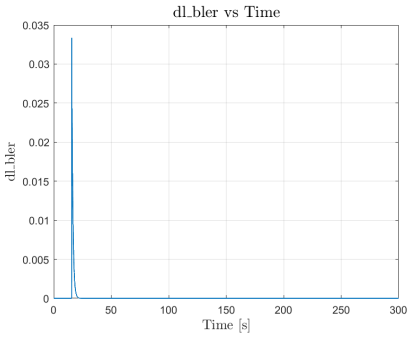
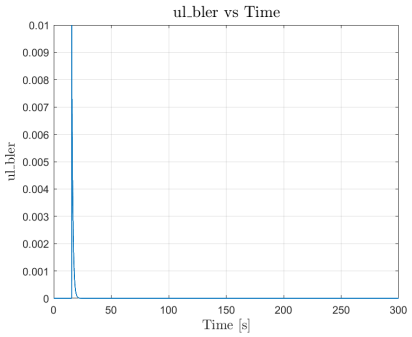


Figure 5.18: RTT in case of robot application deployment

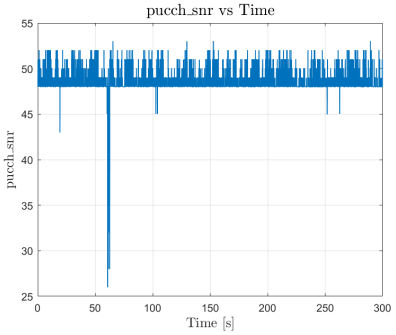
Another comparison was made considering the waiting time and the buffer occupation. These parameters are particularly related to latency, as a high waiting time leads to increased latency, and an occupied buffer can increase latency if the data cannot be transmitted quickly enough.
Once again is it noticed that when the application is fully deployed, the conditions suffer a worsening. The average waiting time goes from 3.1143 us in the first case to 19.1863 us in the second one, with also more frequent and taller peaks. Also the spikes in the buffer occupation increase significantly both in number of and values when considering the deployed application.



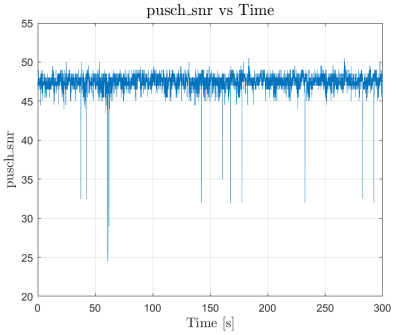
(a) dl_bler



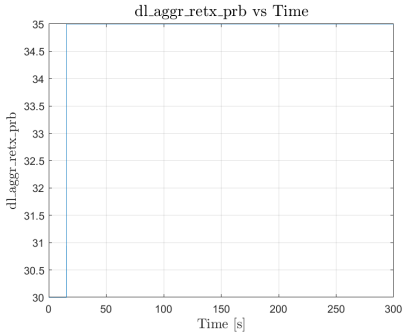
(b) ul_bler



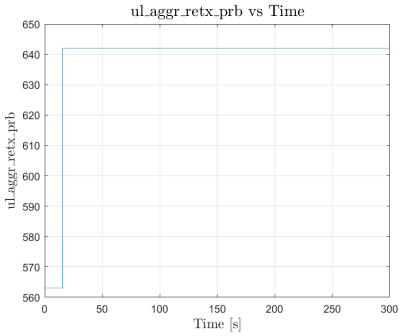
(c) pucch_snr_no_sim



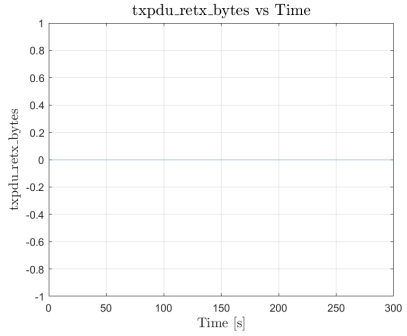
(d) pusch_snr_no_sim



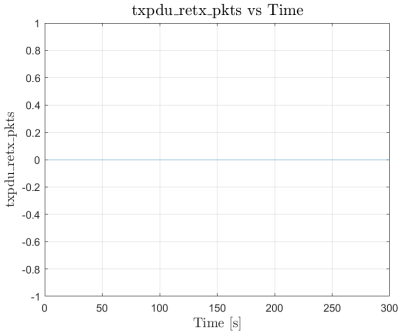
(e) dl_aggr_retx_prb_no_sim



(f) ul_aggr_retx_prb_no_sim

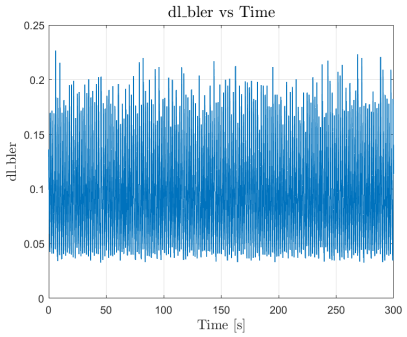


(g) retx_bytes_no_sim

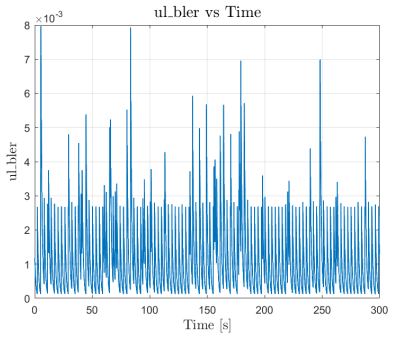


(h) retx_pkts_no_sim

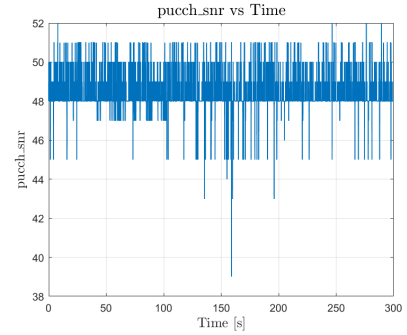
Figure 5.15: KPIs with no simulator running



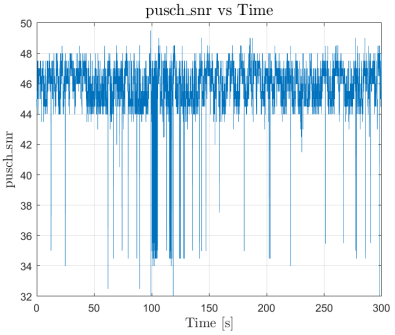
(a) dl_bler_robot



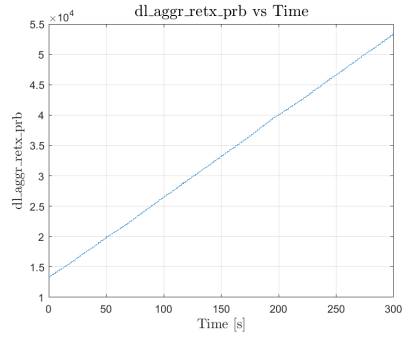
(b) ul_bler_robot



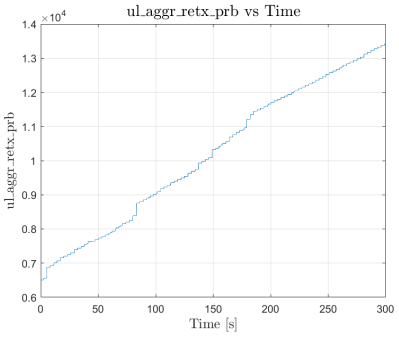
(c) pucch_snr_robot



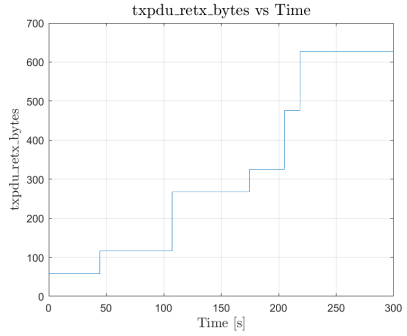
(d) pusch_snr_robot



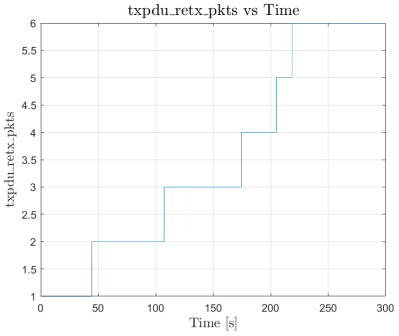
(e) dl_aggr_retx_prb_robot



(f) ul_aggr_retx_prb_robot

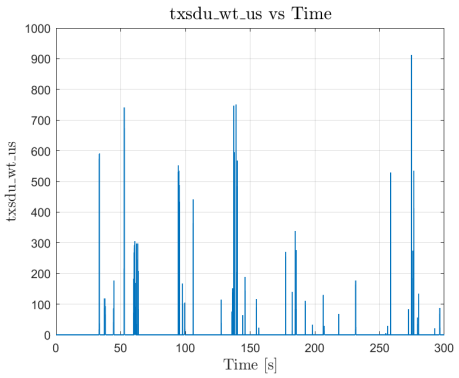


(g) retx_bytes_robot

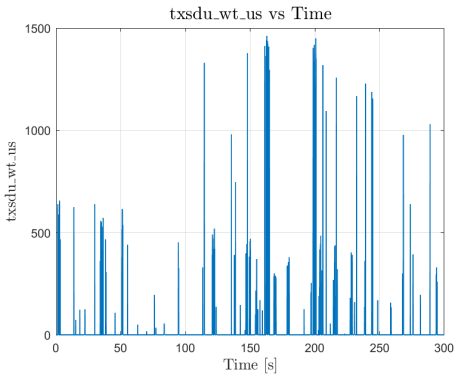


(h) retx_pkts_robot

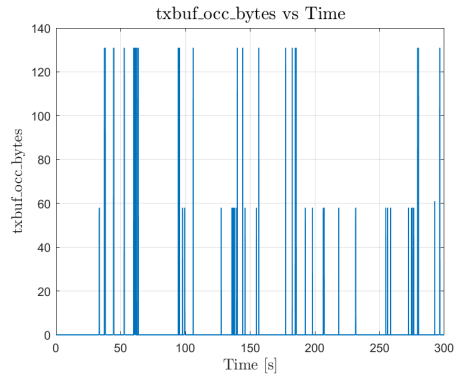
Figure 5.17: KPIs with Gazebo simulator running



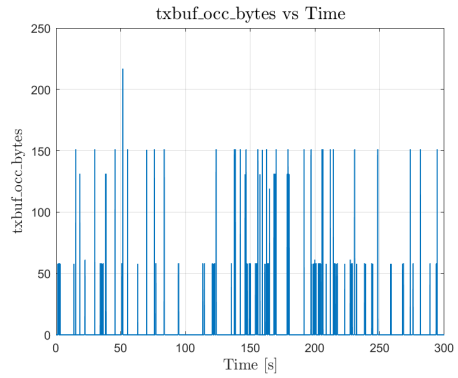
(a) Waiting time no sim



(b) Waiting time robot



(c) Buffer occupation no sim



(d) Buffer occupation robot

Figure 5.19: Waiting time and buffer occupation

Chapter 6

Conclusions

Nowadays Industry 4.0 is becoming increasingly important and studied, thanks to the benefits of its high degree of automation. A common example of this new industrial revolution is automated robots.
A key aspect of this set of applications is low latency, which is fundamental to ensuring the safety of the robot itself and the eventual human workers.
This thesis work showed that it is possible to successfully deploy a robot controller application across a low-cost 5G network, allowing it to receive and send data with low latency, fitting the use case requirement.
The tools exploited in this work are OAI for the network and ROS 2 for the robot.
The main technical limitations encountered are related to the poor computing power of the PCs, the emission power of our antennas, and also the environment in which the experiments took place could have been affected by noise and interference.
Due to the scope of the thesis and the setup of the laboratory environment, the implemented use case is simplified, meaning that certain differences emerge in comparison to a real industrial scenario: a real industrial robot would operate further away from the gNB leading to worse channel conditions and consequent retransmissions, moreover, in a real-world scenario, the robot should manage more complex situations like SLAM and collision avoidance that require a more complex control law and connection reliability. Still, the overall architecture and the logic behind this work have been validated.
This thesis work achieved a good automated trajectory control of a robot across the 5G network, satisfying the latency and throughput requirements.
As a first step, I set up the 5G network overcoming the lack of documentation on OAI, and verified its functionality. Then, I chose the robot model to fit the thesis aim, and I also modified its plugins to make it possible to collect and process the pose data. Successively, I wrote the script to control the robot while also choosing the best possible path to be followed and deployed it across the network I had previously set up.

Finally, I analyzed the system's limitations and studied the correlation between the parameters of the robot and the one of the network.

6.1 Future works

It would be possible to improve the results of this thesis by:

- Upgrade to more powerful laptops to better handle network traffic
- Implement a more reliable network using robust devices and a more stable platform than OAI
- Utilize a more agile robot model paired with a more complex control law

Other improvements may regard network slicing, as a high-reliability slice could provide better channel conditions for the transmission.
Commercial 5G base stations could be used to achieve better results. In fact, during this thesis work, it was not allowed to generate a 5G signal strong enough to interfere with the commercial 5G.

Appendix A

Codes

A.1 gNB MEP configuration file

```
version: '3.8'
services:
    oai-gnb:
        image: oaisoftwarealliance/oai-gnb:mep-compatible
        privileged: true
        container_name: oai-gnb-usrp
        environment:
            RFSIMULATOR: server
            USE_SA_TDD_MONO: 'yes'
            GNB_NAME: gnb
            USE_B2XX: 'yes'  #only needed when using B210
            USE_VOLUMED_CONF: 'yes' #only needed when mounting the configuration file
            TAC: 1
            MCC: '208'
            MNC: '99'
            MNC_LENGTH: 2
            NSSAI_SST: 1
            AMF_IP_ADDRESS: 192.168.70.132
            GNB_NGA_IF_NAME: demo-oai
            GNB_NGA_IP_ADDRESS: 192.168.70.160
            GNB_NGU_IF_NAME: cn5g-access
            GNB_NGU_IP_ADDRESS: 192.168.72.160
            SDR_ADDRS: serial=314BCFF #substituted serial of USRP
            USE_ADDITIONAL_OPTIONS: --sa -E --continuous-tx --log_config.global_log_options level,nocolor,time,line_num,function #(B210)
        volumes:
            - shared_lib:/usr/local/lib/flexric/
```

```
27              - ./conf/flexric.conf:/usr/local/etc/flexric/flexric.conf
28              -  /dev/bus/usb/:/dev/bus/usb/ #(B210)
29              - ./gnb-n78.conf:/opt/oai-gnb/etc/mounted.conf #(B210)
    config file should be present
30          networks:
31              public_net:
32                  ipv4_address: 192.168.70.160
33              public_net_access:
34                  ipv4_address: 192.168.72.160
35          healthcheck:
36              test: /bin/bash -c "pgrep nr-softmodem"
37              interval: 10s
38              timeout: 5s
39              retries: 5
40  ...
```

A.2 ROS 2 plugin

```
1 <plugin
2     filename="ignition-gazebo-pose-publisher-system"
3     name="gz::sim::systems::PosePublisher">
4     <publish_link_pose>false</publish_link_pose>
5     <publish_collision_pose>false</publish_collision_pose>
6     <publish_visual_pose>false</publish_visual_pose>
7     <publish_nested_model_pose>true</publish_nested_model_pose>
8 </plugin>
```

A.3 Gazebo world settings

```
1 <physics type="ode">
2       <max_step_size>0.001</max_step_size>
3       <real_time_factor>1</real_time_factor>
4       <real_time_update_rate>1000</real_time_update_rate>
5 </physics>
```

Bibliography

[1] 3GPP. URL: https://www.3gpp.org/technologies/5g-system-overview (cit. on p. 4).

[2] 3GPP. URL: https://www.3gpp.org/technologies/edge-computing (cit. on p. 6).

[3] Ettus Research. URL: https://www.ettus.com/wp-content/uploads/2019/01/b200-b210_spec_sheet.pdf (cit. on p. 6).

[4] OAI. URL: https://gitlab.eurecom.fr/oai/openairinterface5g/-/blob/develop/doc/NR_SA_Tutorial_OAI_nrUE.md (cit. on p. 7).

[5] OAI. URL: https://gitlab.eurecom.fr/oai/orchestration/blueprints/-/blob/master/mep/README.md (cit. on p. 12).

[6] Steven Macenski, Tully Foote, Brian Gerkey, Chris Lalancette, and William Woodall. «Robot Operating System 2: Design, architecture, and uses in the wild». In: *Science Robotics* 7.66 (2022), eabm6074. DOI: 10.1126/scirobotics.abm6074. URL: https://www.science.org/doi/abs/10.1126/scirobotics.abm6074 (cit. on p. 16).

[7] OpenRobotics. URL: https://docs.ros.org/en/humble/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Nodes/Understanding-ROS2-Nodes.html (cit. on p. 17).

[8] Giuseppe Oriolo, Andrea De Luca, and Marilena Vendittelli. «WMR control via dynamic feedback linearization: Design, implementation, and experimental validation». In: *Control Systems Technology, IEEE Transactions on* 10 (Dec. 2002), pp. 835–852. DOI: 10.1109/TCST.2002.804116 (cit. on p. 19).

[9] OpenRobotics. URL: https://app.gazebosim.org/OpenRobotics/fuel/models/CORO_ROCKY_SENSOR_CONFIG_3 (cit. on p. 20).

[10] DDS Foundation. URL: https://www.dds-foundation.org/how-dds-works/ (cit. on p. 27).

[11] OpenRobotics. URL: https://docs.ros.org/en/humble/Concepts/Intermediate/About-Different-Middleware-Vendors.html (cit. on p. 27).

[12] Eclipse Fundation. URL: https://zenoh.io/blog/2021-04-28-ros2-integration/ (cit. on p. 28).

[13] Robert Schmidt, Mikel Irazabal, and Navid Nikaein. «FlexRIC: An SDK for next-generation SD-RANs». In: *CONEXT 2021, 17th International Conference on Emerging Networking EXperiments and Technologies, 7-10 December 2021, Munich, Germany (Virtual Conference)*. Ed. by ACM. Munich, 2021. DOI: 10.1145/3485983.3494870 (cit. on p. 41).

[14] O-RAN Alliance. URL: https://www.o-ran.org/ (cit. on p. 42).

[15] O-RAN Alliance. URL: https://public.o-ran.org/display/WG3/Introduction (cit. on p. 42).

[16] O-RAN Alliance. URL: https://mediastorage.o-ran.org/white-papers/O-RAN.WG1.Use-Cases-and-Deployment-Scenarios-White-Paper-2020-02.pdf (cit. on p. 42).

[17] OAI. URL: https://gitlab.eurecom.fr/oai/openairinterface5g/-/blob/develop/openair2/E2AP/README.md (cit. on p. 42).

Acknowledgements

Voglio iniziare ringraziando la Professoressa Chiasserini e il correlatore Corrado Puligheddu per avermi seguito nell'attività di tesi. Un ringraziamento anche a Rex; senza i suoi consigli e la sua pazienza non sarebbe stato possibile svolgere questo lavoro.

Ringrazio i miei genitori per avermi sempre spronato a dare il meglio e per averci messo i soldi.
Ringrazio i miei fratelli, anche se non saprei per cosa dovrei ringraziarli visto che non hanno mai fatto nulla per per (soprattutto Marco). Ringrazio per davvero Francesca, che si è sempre dimostrata disponibile per qualsiasi dubbio io avessi.

Ringrazio i miei amici di Torino per aver fatto si che questo percorso universitario fosse ricco di momenti spensierati e divertenti. Un grazie particolare va a Lorenzo, con il quale ho condiviso le gioie e i dolori di questo lavoro di tesi.

Il ringraziamento più grande va a Coco, l'unica che ha sempre creduto in me e non mi ha mai fatto mancare il suo affetto, riuscendo a rendere felice ogni giorno insieme.

Infine, ringrazio Daniela perchè se no si offende. La ringrazio perchè ha reso straordinario l'ordinario, rendendo speciale ogni giorno. Mi è stata accanto e ha creduto in me anche quando io stesso non ci credevo fino in fondo. Grazie alla mia *luce negli occhi*.