Universidad de Oviedo
Universidá d'Uviéu
University of Oviedo

ESCUELA POLITÉCNICA DE INGENIERÍA DE GIJÓN

MÁSTER UNIVERSITARIO EN INGENIERÍA DE TELECOMUNICACIÓN

ÁREA DE TEORÍA DE LA SEÑAL Y COMUNICACIONES

TRABAJO FIN DE MÁSTER

Wireless Control and Coordination of Industrial Robots using 5G

(Control inalámbrico y coordinación de robots industriales mediante 5G)

Villanueva Fernández, Miguel

TUTOR: Rodriguez Larrad, Ignacio

CO-TUTOR: Mogensen, Preben

FECHA: Julio de 2024

Abstract

Rapid technological advancements across many different fields like wireless communications, Internet of Things, cloud computing, robotics, and smart production processes have driven the necessary innovation to make the Industry 4.0 a reality. All of them allow for increased efficiency and safety, and more flexible manufacturing processes. This work aims to investigate, design, deploy and validate an industrial use case for Ultra Reliable Low Latency Communications (URLLC) over 5G, including real-time multi-robot coordination. The selected use case scenario aims at enhancing a traditional production cell composed by two robotic arms, by enabling coordinated mobility of one of them and overall synchronized actuation, resulting in a more flexible and efficient setup.

In order to implement a solution capable of this, three different objectives were set. The first one was to investigate and characterize the time synchronization in 5G networks considering static and mobile use cases with common synchronization mechanisms such as NTP. The next one was to design and develop the industrial testbed over 5G with coordinated wireless robots. The last objective was to validate the use case scenario and evaluate its performance over 5G, comparing it with WiFi 6E as an alternative wireless access method.

All objectives were successfully met. The achieved time synchronization over 5G was below +/-0.1 ms, and the industrial use case was feasible. It was demonstrated that the developed industrial use case involving 5G-coordinated robotic elements had a reliable performance in all runs, with an Operational Closed Loop Latency of 14.33 ms in average, and 19.89 ms as maximum. WiFi 6E in ideal controlled conditions managed to achieve even better results, with a mean of 6.82 ms and a maximum of 9.37 ms. However, as WiFi does not have a scheduled access to the medium, is susceptible of performance degradation if the network is loaded with background traffic. In that scenario, the results showed that the total latency over WiFi 6E was worse than in the 5G case, reaching cycle times of up to 100 ms, which are not acceptable for our URLLC case, where the maximum delay tolerance was 30 ms.

In summary, while WiFi 6E in ideal conditions presented the best performance for the implemented use case, in operational conditions 5G offered the best reliable performance.

Resumen

Los rápidos avances tecnológicos en campos como las comunicaciones inalámbricas, el Internet de las cosas, el cloud computing, la robótica y los procesos de producción inteligentes, han impulsado la innovación necesaria para hacer realidad la Industria 4.0. Todos ellos permiten una mayor eficiencia y seguridad, y procesos de fabricación más flexibles. Este trabajo tiene como objetivo investigar, diseñar, implementar y validar un caso de uso industrial para comunicaciones ultra confiables de baja latencia (URLLC) sobre 5G, incluyendo coordinacion multi-robot en tiempo real. El caso de uso seleccionado intentaría mejorar una célula de producción tradicional compuesta por dos brazos roboticos, permitiendo la movilidad coordinada en uno de ellos y la sincronización temporal en su actuación, resultando en una configuración más flexible y eficiente.

Para desarrollar una solución capaz de ello, se plantearon tres objetivos diferentes. El primero fue investigar y caracterizar la sincronización temporal en redes 5G considerando escenarios estáticos y móviles con mecanismos comunes como NTP. El siguiente fue diseñar y desarrollar el caso de uso industrial sobre 5G con robots inalámbricos coordinados. El último objetivo fue validar el escenario de caso de uso y evaluar su rendimiento sobre 5G, comparándolo con WiFi 6E como método alternativo de acceso inalámbrico.

Todos los objetivos fueron alcanzados. Se consiguió una sincronización temporal sobre 5G inferior a +/-0,1 ms y el caso de uso industrial propuesto fue realizable. Se demostró que el caso de uso industrial desarrollado involucrando elementos roboticos coordinados sobre 5G tuvo un rendimiento confiable en todas las ejecuciones sobre 5G, con una latencia operativa de circuito cerrado de 14,33 ms en promedio y 19,89 ms como máximo. WiFi 6E en condiciones ideales controladas consiguió aún mejores resultados, con una media de 6,82 ms y un máximo de 9,37 ms. Sin embargo, como WiFi no tiene un acceso programado al medio, es susceptible de degradación del rendimiento si la red se carga con tráfico de fondo. En este escenario, la latencia total sobre WiFi 6E solo era peor que en el caso de 5G, alcanzando tiempos de ciclo de 100 ms, completamente inaceptables para un caso de uso de URLLC donde el máximo retardo tolerable es de 30 ms.

En resumen, aunque WiFi 6E en condiciones ideales presentó el mejor rendimiento para el caso de uso implementado, en condiciones operacionales 5G ofrece la mejor confiabilidad.

Contents

1 Introduction **1**

1.1 Motivation 2

1.2 State of the Art 4

1.2.1 Wired/Ethernet networks 4

1.2.2 Wireless industrial networks 5

1.3 Thesis application scenario 6

1.4 Objectives 8

1.5 Thesis outline 9

2 Project Fundamentals **11**

2.1 Data communication 11

2.1.1 5G 11

2.1.2 WiFi 6/6E 13

2.1.3 OSI model 15

2.2 Hardware 17

2.2.1 5G network 18

2.2.2 Wireless gateway 19

2.2.3 UR robotic arms 21

2.2.4 MiR robots 22

2.2.5 ER robot 23

3 Time Synchronization in industrial 5G systems 26

3.1 Introduction 26

3.2 Network Time Protocol 27

3.2.1 Chrony 28

3.3 Robust NTP synchronization in industrial 5G 28

3.3.1 NTP server configuration 30

3.3.2 Base NTP client configuration 31

3.3.3 Base NTP client measurements 32

3.3.4 Optimized NTP client configuration 32

3.3.5 Optimized NTP client measurements 34

3.4 Time synchronization conclusions 36

4 Design and Development of an Industrial Testbed with Coordinated Robots 38

4.1 General requirements 38

4.2 Physical architecture 41

4.3 Software architecture 42

4.3.1 5G gateway configuration for IT/OT integration 42

4.3.2 Static UR robot 45

4.3.3 Mobile ER robot (MiR+UR) 48

4.3.4 Master Controller 50

4.3.5 MQTT broker implementation 51

4.3.6 Message sequence diagrams 52

4.4 Operational KPIs for the demonstration execution 54

4.5 Industrial Testbed Development Conclusions 56

5 Validation and Performance Evaluation Results 57

5.1 Validation 57

5.2 Industrial use case performance evaluation results over 5G 63

5.2.1 KPI1: Operational Closed Control Loop Latency 63

5.2.2 KPI2: Percentage of Successful Iterations 66

5.3 Industrial use case performance evaluation results over WiFi 6E 67

5.3.1 KPI1: Operational Closed Control Loop Latency 69

5.3.2 KPI2: Percentage of Successful Iterations 74

5.4 Validation and Performance Evaluation Conclusions 74

6 Conclusions and future work 76

6.1 Conclusions 76

6.2 Future work 77

6.2.1 Better time synchronization 77

6.2.2 Centralised controller functions 77

6.2.3 Intel RealSense camera streaming 78

A Code developed for the Use Case Implementation 87

A.1 mqtt_Slave_static.py 87

A.2 mqtt_Slave_mobile.py 96

A.3 mqtt_Slave_controler.py 107

A.4 my_tools.py 120

A.5 consts.py 122

A.6 mir_api_s.py 125

List of Figures

1.1 Examples of emerging technologies [1] 1

1.2 History of the industrial revolutions [2] 3

1.3 Typical wired industrial PLC in a factory [6] 5

1.4 Possible use cases for the interconnected Industry 4.0 [12] 6

1.5 Traditional industrial multi-robotic cell scenario 7

1.6 Proposed industrial multi-robotic cell wireless scenario over 5G 8

2.1 5G use cases: eMBB, URLLC and mMTC [15] 12

2.2 Representation of the problem of mobility and handover in WiFi 14

2.3 OSI layers architecture 15

2.4 MQTT generic architecture diagram [30] 16

2.5 AAU 5G Smart Production Lab 18

2.6 Private 5G infrastructure at the AAU 5G Smart Production Lab 19

2.7 5G to Ethernet gateway SBC from the top and bottom 20

2.8 SIM8262E-M2 5G modem 20

2.9 UR5e and UR5 robotic arms 21

2.10 MiR 100 robot and its graphical control webpage 22

2.11 MiR 100 robot and its graphical control webpage 23

2.12 Block programming of the ER robots [41] 24

2.13 ER robot picking up a tray using a checkerboard marker as positional reference . 25

3.1 NTP Stratum architecture [51] 27

3.2 NTP architecture for 5G synchronization testing 29

3.3 Baseline 5G NTP synchronization accuracy for the different reference cases . . . 33

3.4 Optimized 5G NTP synchronization results as compared to the baseline ones . . 35

3.5 Statistical representation of the optimized 5G NTP syncrhonization results as compared to the baseline ones 36

4.1 Sketch of the proposed use case idea 39

4.2 Demo physical architecture 41

4.3 SIM card and 5G modem connection in one of the 5G gateways 43

4.4 Laptop connected to the configuration port of the 5G router 43

4.5 FireHOL configuration 45

4.6 UR robot software architecture 46

4.7 Mobile ER robot software architecture 48

4.8 MiR-Cloud-UR message sequence diagram 53

4.9 UR-Cloud-UR message sequence diagram 54

4.10 Evaluation of recorded traffic captures with Wireshark 55

5.1 MiR robot in Ready state (green lights) 57

5.2 Chrony NTP synchronization query 58

5.3 Demo images preparation steps 59

5.4 Demo images initial steps previous to the gripper release 60

5.5 Demo images from the gripper release event 61

5.6 Demo images finishing steps 62

5.7 Uplink latency CDF graph (representative of Ttx) 63

5.8 Downlink latency CDF graph (representative of Trx) 64

5.9 Processing time in the cloud CDF graph (representative of Tpr) 65

5.10 Operational Closed Control Loop Latency CDF graph (OCCLL) 66

5.11 WiFi network architecture diagram 67

5.12 Comparison of 5G and WiFi NTP synchronization 68

5.13 Uplink latency CDF graph of all scenarios 70

5.14 Downlink CDF graph all scenarios 71

5.15 Processing time in the cloud CDF graph all scenarios 72

5.16 Operational Closed Control Loop Latency CDF graph all scenarios 74

6.1 Centralised controller MiR-Cloud-UR message sequence diagram 78

List of Tables

5.1 Uplink time statistics (Ttx) reference values 69

5.2 Downlink time (Trx) reference values 71

5.3 Processing time in the Cloud (Tpr) reference values 72

5.4 Operational Closed Control Loop Latency (OCCLL) reference values 73

1. Introduction

Technology is advancing at a tremendous pace. It has revolutionized the way we live, work, and interact with the world around us. Rapid advancements across many different fields have driven innovation, transforming industries and shaping the present, and the future, of how society works. In the last decades, we have experienced an unprecedented wave of technological achievements, mainly in the fields of telecommunications and computing. Some of them can be observed in Artificial Intelligence/Machine Learning (AI/ML), Wireless communications, Virtual Reality, the Internet of Things (IoT), and many more as depicted in Figure 1.1. These advancements have not only enhanced our capabilities but have also presented new opportunities and challenges.

AI Engineering
Zero Trust & Cybersecurity
Distributed Cloud & Edge Computing
Privacy Computation
Remote & Digital Workspace Technologies
5G
Human-centered Digital Experience
Intelligent Business Processes & Vendor Platforms
RPA & Hyperautomation
Internet of Behaviors (IoB)

Figure 1.1.- Examples of emerging technologies [1]

These advancements have significantly changed our daily lives, providing us with smarter services and tools that feel more intuitive, responsive, and immersive. In particular, one of the most benefited applications in this regard has been the industry. Machines are generating more data than ever, and in exchange they can be directed with a finer control, and they can even implement new functionalities such as failure prediction systems as means of prevention.

Behind these actions there are predictive models, smart algorithms, and autonomous systems that continually evolve and optimize their performance. All these different technologies and techniques are what nowadays compose the Industry 4.0. Within this context, this work will assess a number of relevant technologies and explore advancements within relevant Industry 4.0 scenarios and applications.

1.1.- Motivation

But why exactly do we need these advancements that come with the Industry 4.0? There is one specific target that the industry has been aiming since it was born, and that is automation, because with it comes the ability to produce large quantities of products in an efficient and profitable way. The evolution of industry from the first industrial revolution to the current state has been marked by significant technological advancements and paradigm shifts in manufacturing and production processes, as it is graphically summarized in Figure 1.2.

The first industrial revolution started in the late 18th century in England, and it was characterized by the mechanization of production processes. These were primarily driven by the invention of the steam engine and several machine tools, and the mechanization of textile manufacturing.

Around a century later started the second industrial revolution, which was marked by advancements in mass production and assembly-line manufacturing techniques. Key innovations included the introduction of electricity, the assembly line, and interchangeable parts. This led to significant improvements in productivity, efficiency, and standardization, particularly in industries like automotive and steel production.

The third industrial revolution, also known as the Digital Revolution, was characterized by the widespread adoption of electronics, computers, and automation technologies such as PLCs that began to take place at the end of the 20th century. This revolution led to increased automation of manufacturing processes, improved precision, and the emergence of new industries such as electronics and telecommunications.

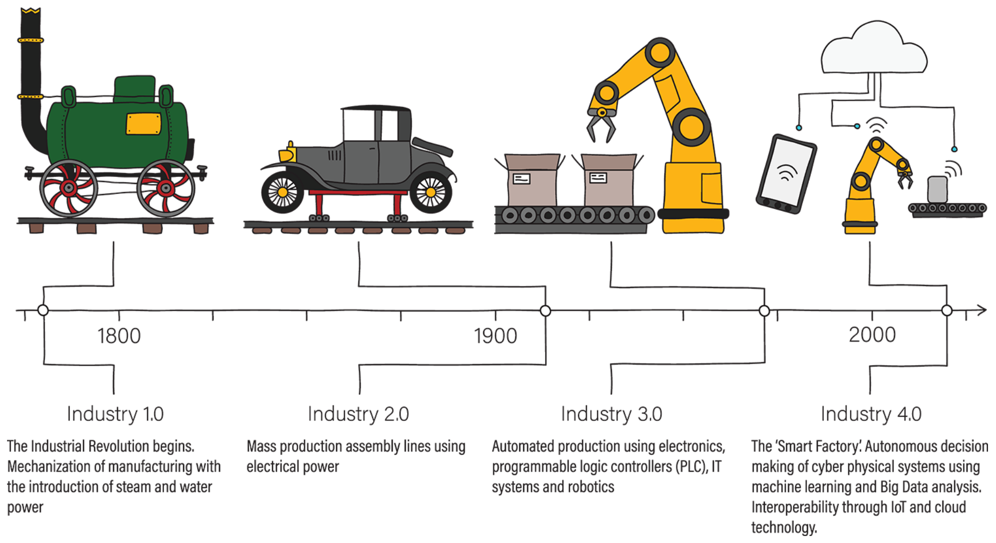

1800
1900
2000
Industry 1.0
The Industrial Revolution begins. Mechanization of manufacturing with the introduction of steam and water power
Industry 2.0
Mass production assembly lines using electrical power
Industry 3.0
Automated production using electronics, programmable logic controllers (PLC), IT systems and robotics
Industry 4.0
The 'Smart Factory'. Autonomous decision making of cyber physical systems using machine learning and Big Data analysis. Interoperability through IoT and cloud technology.

Figure 1.2.- History of the industrial revolutions [2]

And at last, the present state. The fourth industrial revolution, or Industry 4.0, is taking place, and it builds upon the digital revolution with the integration of advanced communication systems, Cyber-Physical Systems (CPS), and the Internet of Things (IoT). Some key technologies include wireless networks, AI/ML, big data analytics, cloud computing, robotics, and IoT sensors [3]. All of them enable the creation of ”smart factories” where machines, products, and systems communicate and cooperate with each other autonomously, leading to many benefits:

- Highly flexible manufacturing processes: using wireless networks to communicate the machines in the factory opens the possibility of having fast and easily reconfigurable production cells and mobile robots that lead to increased flexibility and efficiency.
- Innovation and growth: Industry 4.0 offers opportunities for innovation and new business models. Companies can develop and commercialize novel products and services enabled by these cutting-edge technologies, opening up new revenue streams and driving growth by exploiting competitive advantages.
- Increased quality and efficiency: combining all Industry 4.0 technologies, such IoT, AI/ML, robotics, and advanced data analytics, processes can be streamlined and optimized in efficiency. Companies are motivated to adopt these technologies to reduce costs and improve productivity.

- Safety and reliability: more sensors help identify potential failures and mitigate safety risks with predictive maintenance. In addition, they enhance safety training and awareness, and create safer work environments for employees.

1.2.- State of the Art

Industrial network communications have evolved greatly in the last decades. The overall trend has been to move from cabled electric buses towards Ethernet-based networks; which have been more recently complemented by wireless network systems such as Wi-Fi or 5G [4].

1.2.1.- Wired/Ethernet networks

Once Programmable Logic Controllers (PLCs) were starting to become prevalent in the industry, a way of communication between them was needed. At first, this was done through dedicated automation networks called ”fieldbus systems”, that had to integrate communications at a physical level with CAN, PROFIBUS or INTERBUS networks.

As Ethernet got increasingly popular in the Information Technology (IT) world, it started to be adopted by the industry, even though it lacked genuine real-time capabilities in its standard form, but it brought high-speed data transmission and seamless integration with many different types of networks, unlike previous fieldbus technologies. Many dedicated solutions arose to solve those problems, and even some standards were developed for Real-Time Ethernet (RTE), like PROFINET-Isochronous Real Time (IRT) or EtherCAT [5]. This way, Ethernet enables real-time communication between devices such as PLCs, human-machine interfaces (HMIs), sensors, and actuators at sub-millisecond speeds.

The main problem with the use of this technology is that, being cabled, it is a fixed solution. Mobility is heavily restricted, and set up complexity increases exponentially as the number of devices grow, as it is displayed in the photo of Figure 1.3 where just a few devices are connected. That is the reason why, as wireless technologies evolved and increased their capacity and reliability, implementations without cables grew in industrial deployments over the last years.

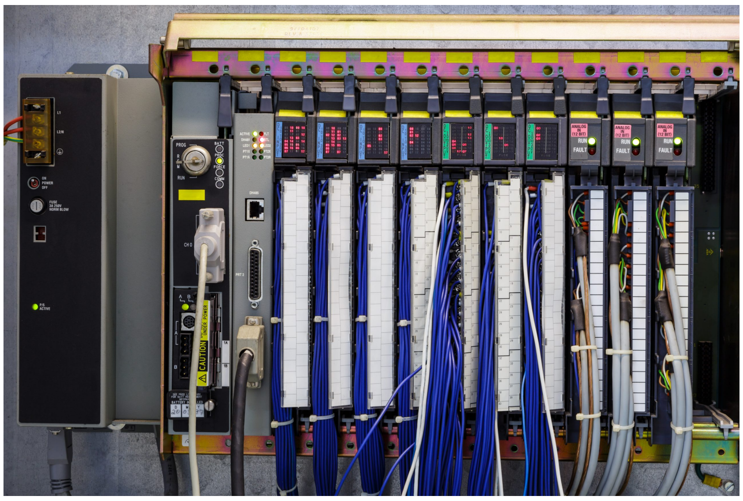


Figure 1.3.- Typical wired industrial PLC in a factory [6]

1.2.2.- Wireless industrial networks

The first means of wireless communications between computers appeared in the 70s, but it was in the 90s that both IEEE 802.11 (also nicknamed WiFi) and the second generation of mobile communications (2G) brought them to the general public. The problem is that, even though they have been increasingly used credit to their flexibility, it was only in subsystems, as they were very far from cabled solutions in terms of reliability for the actual infrastructures. Anyhow, wireless communications have evolved a lot in the recent times, and many new different technologies that could be used in factories have appeared like WirelessHART [7], ZigBee [8] or even LoRa [9]. But both IEEE 802.11 and cellular networks continue being the reference in performance with their latest revisions. Strict requirements in latency, coverage, reliability and big data throughput (in cases like depth or high resolution cameras), render 5G [10] and WiFi 6/6E [11] as the main wireless options for the demanding scenarios inside the factories.

These wireless networks bring many new possible use cases to the Industry 4.0 revolution [13]. Figure 1.4 depicts some of them. First of all, it makes possible the transition from just Autonomous Guided Vehicles (AGVs), that need predefined and fixed routes to work, into Autonomous Mobile Robots (AMRs), that can handle navigating changing environments. This last type robots were not possible before, as they inherently need wireless communications to navigate the changing environments usually found in warehouses, for example. In addition,

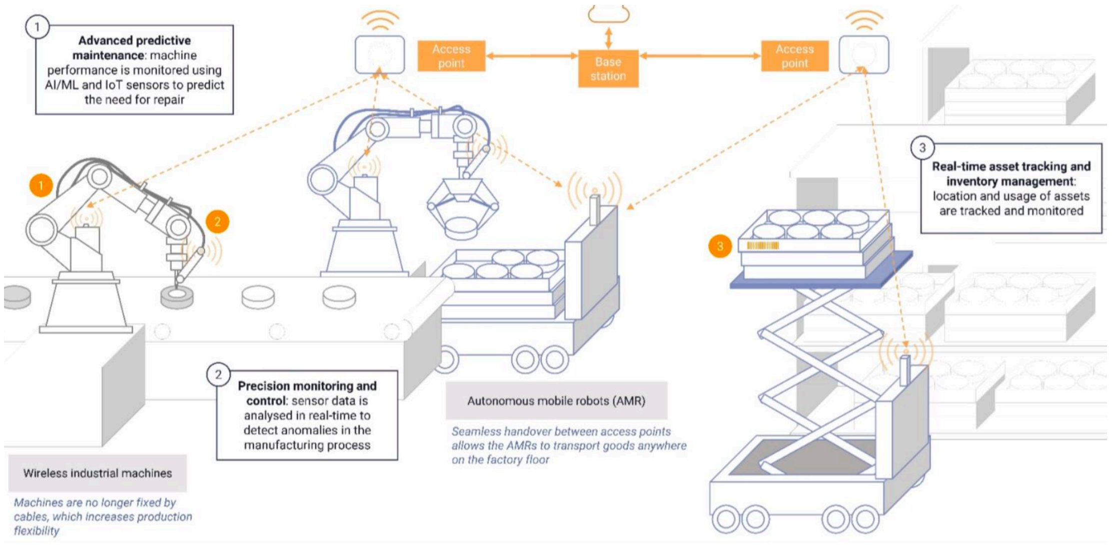

1 Advanced predictive maintenance: machine performance is monitored using AI/ML and IoT sensors to predict the need for repair
Access point
Base station
Access point
3 Real-time asset tracking and inventory management: location and usage of assets are tracked and monitored
2 Precision monitoring and control: sensor data is analysed in real-time to detect anomalies in the manufacturing process
Autonomous mobile robots (AMR)
Seamless handover between access points allows the AMRs to transport goods anywhere on the factory floor
Wireless industrial machines
Machines are no longer fixed by cables, which increases production flexibility

Figure 1.4.- Possible use cases for the interconnected Industry 4.0 [12]

wireless communications can also benefit machines without full mobility by making them very easily movable and re-configurable. Deployments or changes for new functionalities become much faster and cheaper than with cabled solutions. Furthermore, having all these machines interconnected will allow for real-time tracking of assets and inventory, and even the predictive maintenance and anomalies monitoring. Lastly, generating so much data in real-time permits creating digital twins that perfectly simulate the real behaviours and situations, and even Augmented Reality (AR) or Virtual Reality (VR) use cases.

1.3.- Thesis application scenario

Up until the 4th industrial revolution, the traditional industrial scenario would be to have robots in a factory, like a robotic arm with its sensors and actuators, that are driven by PLCs through cable technologies such as fieldbus or Ethernet. This is depicted in Figure 1.5. These PLCs are in the end the controllers that are connected to the different sensors and actuators of a system, such as a robotic arm, and takes the decisions on what to do. These PLCs can be communicated together to coordinate different robots by wired networks and other high level PLCs or computers.

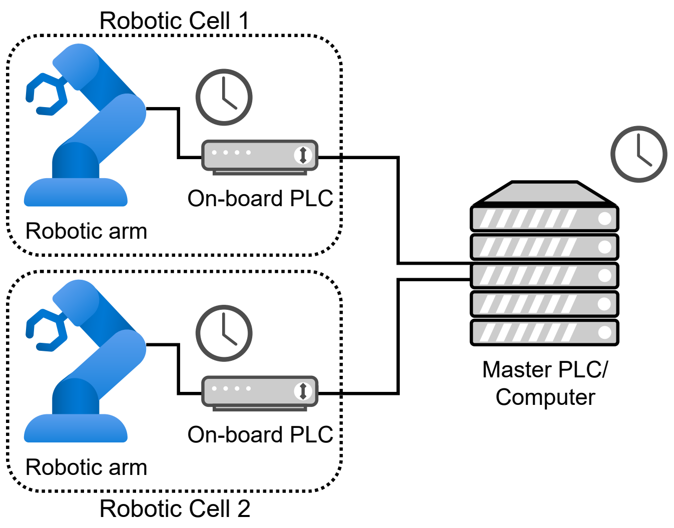


Figure 1.5.- Traditional industrial multi-robotic cell scenario

Meanwhile, nowadays the trend is to create more wireless links to communicate robots. First it was just to extract and analyze data out of the machines to prevent failures or warn about dangers. But as wireless networks became better, the new aim is to also control and coordinate robots over wireless networks, to allow for mobile robots and more easily configurable factories. This involves a more complex architecture, which inevitably leads to worse performance and reliability than wired networks, but that enables new functionalities never possible before.

As depicted in Figure 1.6, the proposed wireless architecture over 5G would be to connect a 5G modem instead of a PLC to the robotic arm. This device would connect to the software PLC that is running in the Edge Cloud of the factory over the 5G network. This network is composed of a 5G Radio Access Network to provide the wireless access to the network, a 5G Core that executes all the 5G functionalities and enables communication with other users, and then the own factory network to connect wired elements and other networks.

In order to execute the movements with precision and accuracy at the right time, each robot needs exact instructions at the right time. But the clocks present in the microprocessors and microcontrollers of these devices are not very precise, that is why there are different time synchronization mechanisms that ensure agreement between nodes in a network. Cabled

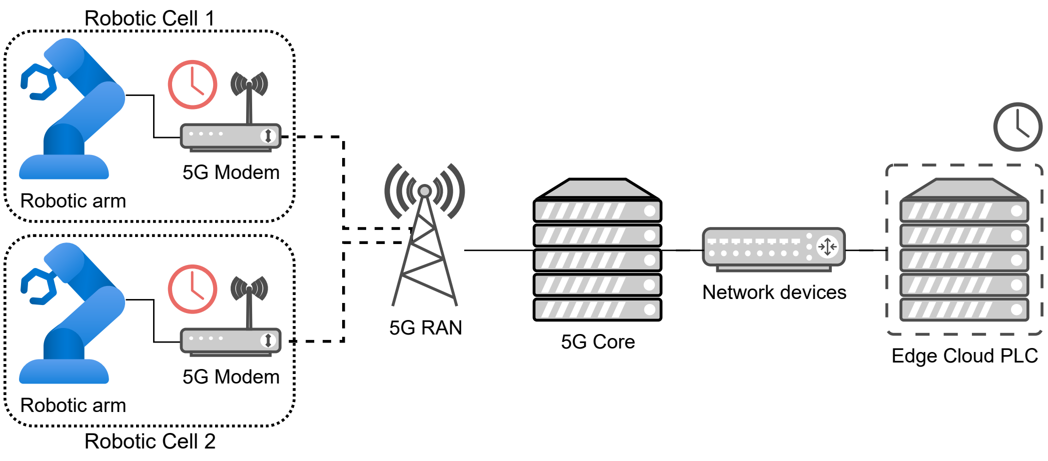


Figure 1.6.- Proposed industrial multi-robotic cell wireless scenario over 5G

solutions are very stable in this regard, as both latency (delay between ends in a message transmission) and jitter (difference in latency between messages) are very consistent and minimal

Meanwhile, time synchronization is more difficult to achieve in wireless networks, as they have more latency and jitter, which can affect the delay of the instructions and the drift in the clocks. Wireless solutions offer many advantages, like enabling full mobility and easily reconfigurable production chains. However, in order to make those scenarios a reality, it is very important to guarantee robot coordination and synchronization, and that requires the study and development of new technologies and techniques.

1.4.- Objectives

In general terms, the global aim with the project is the investigation and validation of wireless architectures and protocols for the synchronization and control of industrial robots. The capabilities of new advanced technologies such as 5G and WiFi 6E will be examined and put in perspective of the limitations from traditional wired networks, taking full advantage of the wireless potential. Both synchronization, latency and possible new functionality are evaluated to investigate the new use cases that these state-of-the-art networks can bring into factories. Within this general objective, some specific action points can be defined:

- OB1: investigation and characterization of time synchronization in 5G networks considering static and mobile industrial use case conditions.
- OB2: design and development of a industrial testbed including 5G wireless-synchronized behavior of robotic entities.
- OB3: validation and performance evaluation of the implemented 5G solution and comparison with alternative wireless access methods (e.g., WiFi 6E).

This work will serve as a reference for the experimentation and implementation of new technologies and use cases in the industrial domain that could benefit from the recent development of wireless networks. In addition, the results can be useful to researchers and engineers in the areas of sensors, IoT, computer networks, wireless communications and different industry verticals making use of robotics, especially in manufacturing.

1.5.- Thesis outline

This section comprises a summary of every chapter in the thesis.

- Chapter 1 - Introduction: it is a general presentation to the motivations of the research, an analysis of the state of industrial networks with the new possible use cases that bring the use of wireless networks in factories, and the objectives set.
- Chapter 2 - Project Fundamentals: it exposes the fundamental concepts needed to understand the used technologies and then specifies the available hardware at the AAU 5G Smart Production Lab to develop the thesis.
- Chapter 3 - Time Synchronization in industrial 5G systems: this chapter explores the tools available to synchronize industrial devices over wireless networks, and their performance both in stock and optimized form.
- Chapter 4 - Design and Development of an Industrial Testbed with Coordinated Robots: it explains the process of selecting, designing and implementing an industrial use case to showcase the possibilities of coordinated robots over 5G.
- Chapter 5 - Validation and Performance Evaluation Results: the resulting demonstration, and its closed control loop in specific, is evaluated in performance over a 5G and a WiFi 6E network.

- Chapter 6 - Conclusions: some conclusions are drawn from the results presented in each chapter, showcasing as well future areas of work and improvement.

2. Project Fundamentals

This chapter surveys the most relevant technologies and physical hardware elements considered in the development of the project.

2.1.- Data communication

In this section, the two main wireless technologies that are being adopted by industry are introduced, along with a brief recapitulation of all the network layers involved for telecommunication, as well as some of the most commonly found protocols.

2.1.1.- 5G

Cellular networks managed to offer wireless packet connectivity with 2G even before WiFi, but it was very limited until 3.5G and 4G (HSPA+ and LTE) were brought along. These technologies managed to provide users speeds closer to the ones delivered by WiFi in the early 2010s, but with a much wider connectivity range. It meant a revolution for the mobile phone world, as many more applications and use cases sprung up.

The fifth generation of cellular technology (also known as 5G), is designed to be a unified, more capable platform that not only elevates today's mobile broadband experiences but also supports new services and capabilities for a wide range of industries. There are three primary scenarios inside 5G, often referred to as 5G use cases, represented in Figure 2.1: Enhanced Mobile Broadband (eMBB), Ultra-Reliable Low Latency Communications (URLLC), and Massive Machine Type Communications (mMTC) [14].

eMBB is the most common scenario, as it is aimed to the human-centric use cases for access to multimedia content, services and data. It aims to improve the performance of the general mobile broadband services. This includes faster data speeds and higher capacity for Augmented and Virtual Reality (AR/VR), High-definition video streaming, and cloud services.

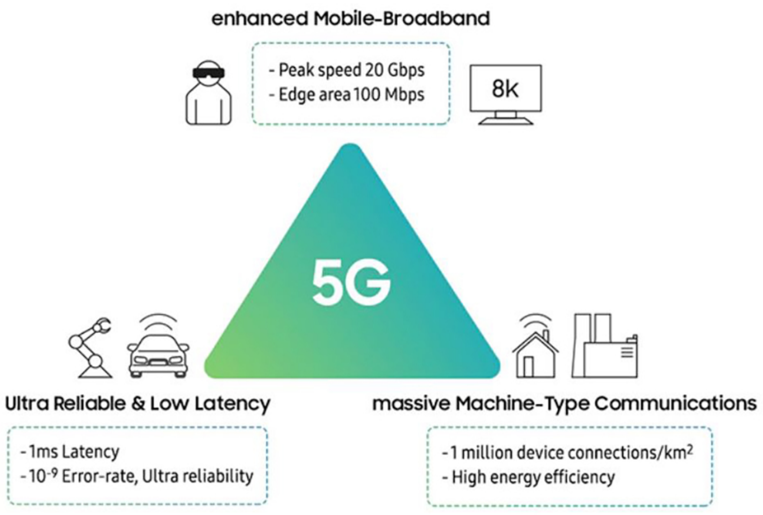

enhanced Mobile-Broadband
- Peak speed 20 Gbps
- Edge area 100 Mbps
8k
5G
Ultra Reliable & Low Latency
- 1ms Latency
- 10^{-9} Error-rate, Ultra reliability
massive Machine-Type Communications
- 1 million device connections/km^2
- High energy efficiency

Figure 2.1.- 5G use cases: eMBB, URLLC and mMTC [15]

URLLC is probably the most related scenario with this thesis, as it is designed for applications that require highly reliable and low-latency communication. This scenario is essential for mission-critical and time-sensitive applications where even minimal delays can have significant consequences. Those applications could be real-time wireless industrial automation, self-driving cars, or even remote surgery.

The last one, mMTC, is more focused on Internet of Things (IoT). It is designed for scenarios with a very large number of connected devices typically transmitting a relatively low volume of non-delay sensitive data. This is best suited for applications like smart cities, where numerous sensors can be deployed throughout a city to monitor and manage resources like water, electricity, and traffic. Smart agriculture is as well a part of it, as sensors can monitor soil moisture, temperature, and other factors to optimize farming practices, and even wearable devives such as fitness trackers, health monitors, and other wearable technology can benefit from consistent, low-power connectivity.

As it was discussed earlier, URLLC is the most useful use case in factories, because the industry needs very high reliability for their processes as they are usually running all the time and they want to minimize downtime. But there is an aspect that can affect the connection in scenarios with mobility, and that is user handover between cells [16]. There are three types of handover ranging from best to worst: Intra-gNodeB Handover, as the User Equipment (UE) does not change of 5G base station; Inter-gNodeB Handover, in which a UE changes from one 5G base station to another; and Inter-RAT (Radio Access Technology) Handover, as the user has to change of radio technology to operate, such as from 5G to 4G. There are many parameters that are monitored to ensure this action is done with as little impact on the UE connection as possible, and 5G improves this with network slicing and with dual connectivity. These technologies allow the user to use an specific network slice depending on its needs, and to be able to connect to both access points and even two different 5G base stations in order to minimize the effect of handover.

In addition, 5G operates over a licensed spectrum, which in Europe it would be over the 3.5 GHz band (n78) for industrial cases (although this could change depending on the legislation of each country). Being a licensed spectrum, the access to the radio-electric medium employs a scheduled mechanism, in which devices wait for time slots to transmit data. This translates to a more reliable and deterministic performance that increases the guarantee of Quality of Service (QoS) [17].

All these technologies make 5G the most desirable wireless network choice for industrial implementations.

2.1.2.- WiFi 6/6E

The first WiFi standard, referred as IEEE 802.11, appeared in 1997, offering up to 2 Mbps of speed [18]. This was quickly improved with ”b” and ”a” revisions, that operated at a maximum of 11 Mbps in the 2.4 Ghz band, and achieving even 54 Mbps at 5 Ghz. Afterwards, steady improvements in the radiocommunication part such as advanced modulation techniques, MIMO and OFDMA managed to steadily improve the maximum speeds to up to 9.6 Gbps in WiFi 6.

All these advancements made WiFi the most common wireless connectivity solution in every kind of scenario, from homes to offices. But while it can meet many application requirements, it is not always ideal, mainly due to its operation in unlicensed bands, leading to potential interference from neighboring networks. WiFi uses mechanisms like Listen-Before-Talk (LBT) to prevent collisions by ensuring a channel is clear before transmitting. This is crucial in the congested 2.4 and 5 GHz bands (and also applies to 6 and 60 GHz) due to widespread use by technologies such as Bluetooth and Zigbee [19]. These methods can introduce latency as devices must wait for a clear channel before transmitting data, specially in areas congested with many of these wireless devices, even if the WiFi network is not crowded. That is the reason why WiFi 6E is the most promising form of all the options for industrial use cases, specially time-sensitive ones, as they will likely experience less adverse effects from it in the 6 GHz band [20].

Another concern when using WiFi in industrial scenarios is mobility of devices and their handover processes between access points, as they differ between those of 5G. In this case, it is handled by the Base Station (BS), which determines when and how to manage roaming as devices move out of one Access Point (AP) into another [21]. During this transition, there is a brief loss of connection because Wi-Fi uses a ”break before make” approach, disconnecting from the first AP before connecting to the second. This can cause devices, such as Autonomous Mobile Robots (AMRs), to temporarily lose connection in the border area between APs, as represented in Figure 2.2. Although advancements in Wi-Fi 6 (IEEE 802.11ax) aim to improve mobility, they still fall short of meeting the tough reliability demands for the industry [22].

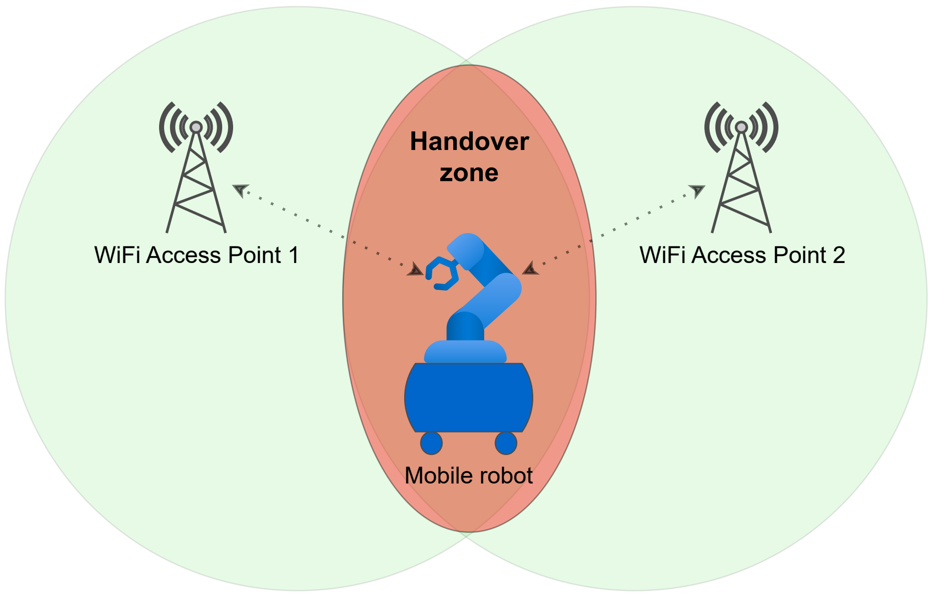


Figure 2.2.- Representation of the problem of mobility and handover in WiFi

2.1.3.- OSI model

According to the Open Systems Interconnection (OSI) model for communications developed by the International Organization for Standardization (ISO) [23], there are seven different layers in the network architecture as depicted in Figure 2.3.

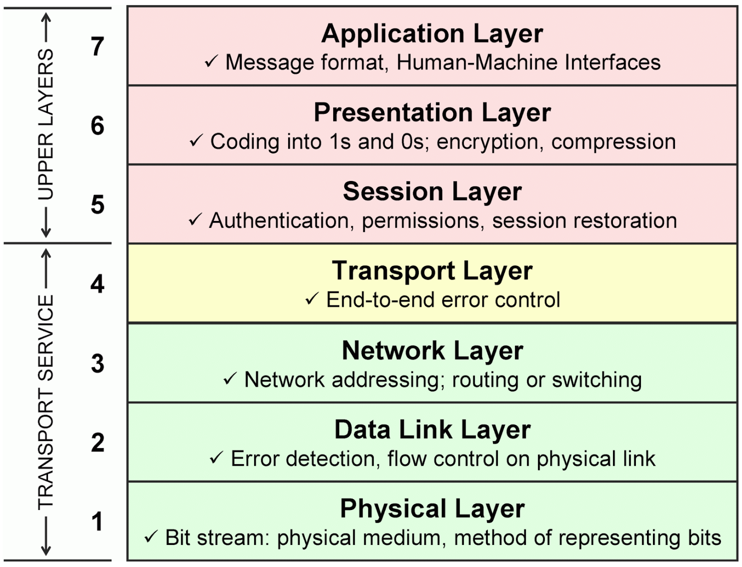

UPPER LAYERS
TRANSPORT SERVICE
7
Application Layer
✓ Message format, Human-Machine Interfaces
6
Presentation Layer
✓ Coding into 1s and 0s; encryption, compression
5
Session Layer
✓ Authentication, permissions, session restoration
4
Transport Layer
✓ End-to-end error control
3
Network Layer
✓ Network addressing; routing or switching
2
Data Link Layer
✓ Error detection, flow control on physical link
1
Physical Layer
✓ Bit stream: physical medium, method of representing bits

Figure 2.3.- OSI layers architecture

The first layer describes the transmission and reception of raw data bits over the physical medium. Therefore, it defines the hardware equipment, wiring, frequencies and signals used. In cabled solutions it would be the Ethernet cable and Network Interface Cards (NICs), but in wireless solutions like WiFi or 5G involves the electromagnetic waves, antennas and modems used to transmit and receive the signals over the air.

Then, the Data Link layer describes how data frames are transferred and handled over the different nodes. This includes error detection and correction, and flow control to ensure integrity in the communications. Medium Access Control (MAC) is a crucial part of this layer, as its major responsibility is to offer reliable link to link data transfer [24].

The Network Layer, the third one, is responsible for data routing, packet forwarding and fragmentation, and logical addressing of devices attached to the network. The Internet Protocol (IP) [25] is in charge of all these functions by giving nodes an IP address, determining the best paths for reaching other computers, controlling the fragmentation and the integrity of the data, and limiting the lifespan of packets to prevent infinite loops. The Internet Control Message Protocol (ICMP) [26] also operates in this layer, offering many diagnostics services, one of them being the ”ping” messages.

Afterwards, in the Transport Layer, the protocols are used to provide end-to-end communication services for applications [27], with the main options being TCP and UDP. TCP is a reliable, connection-oriented transport service that ensures end-to-end reliability, re-sequencing, and flow control. In contrast, UDP is a connectionless transport service, often referred to as ”datagram” service, that is used when quick transmission times are essential in detriment of the reliability.

MQTT is a Client Server publish/subscribe messaging transport protocol that operates above the Transport Layer, and it is designed to be light weight, open, simple, and easy to implement. It is ideal for many situations, including constrained environments such as for communication in Machine to Machine (M2M) and Internet of Things (IoT) contexts where a small code footprint is required and/or network bandwidth is at a premium. The protocol runs over TCP/IP, or over other network protocols that provide ordered, lossless, bi-directional connections [28]. Along with OPC-UA, they are becoming the main options for implementing communications between machines inside factories in the Industry 4.0 [29]. Although OPC-UA offers very interesting features, MQTT is proven to provide superior performance when applied to wireless robotic cells, so it is the best option to develop the robot’s control for this thesis.

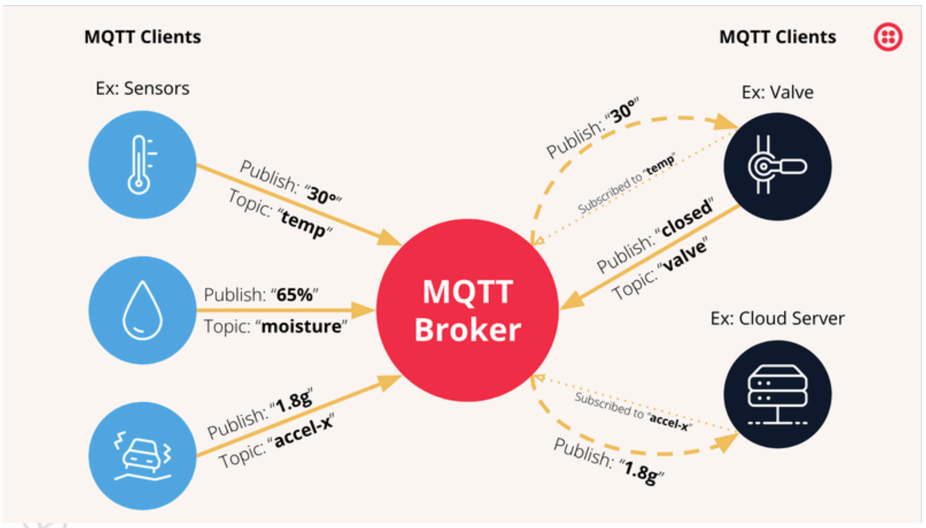


Figure 2.4.- MQTT generic architecture diagram [30]

The general architecture of MQTT is composed by three different nodes, represented in Figure 2.4: the publishers, the message broker and the subscribers. Publishers are the nodes in charge of sending the messages. They categorize their information using topics, and they send it to the broker. Meanwhile, the broker is the intermediary between publishers and subscribers. It forwards data based on their topic subscription. Lastly, the subscribers are the final nodes that receive the messages, so they have to notify the the broker which messages they want to receive using a specific topic (or a wildcard to subscribe to multiple). This architecture makes the communication asynchronous, as the receiver does not need to poll the transmitter for new updated information, making all the interactions more efficient because there are less messages in the network.

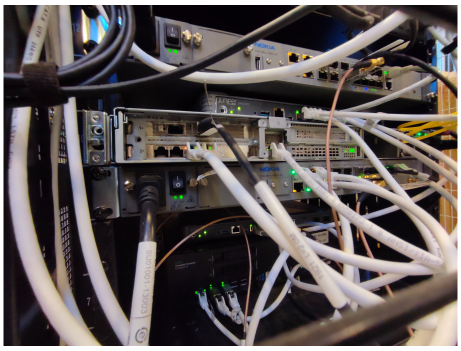
2.2.- Hardware

To give a real-world industrial dimension to the project, it was developed at the AAU 5G Smart Production Lab at Aalborg University, Denmark. Here, a small industrial research facility with a dedicated commercial industrial 5G network with full coverage was available for experimentation. Likewise, real industrial machinery including autonomous mobile robots, robotic arms, production lines, etc, was also at hand for prototype implementation. A picture of the lab is shown in Figure 2.5.

Figure 2.5.- AAU 5G Smart Production Lab

2.2.1.- 5G network

It is out of scope for this thesis to explain in detail how the complex combination of 4G, 5G NSA, 5G SA, WiFi 5 and WiFi 6 operate all together inside the AAU 5G Smart Production Lab [31]. That is why only the 5G Stand Alone private network will be described. The equipment for that part of the network comes from Nokia, and is operated at the laboratory site. Is is composed of a private 5G NR mini-core with a pico Base Station and three Access Points, pictured in Figure 2.6. The radio access part serves connection on the 3.7 GHz band in Stand Alone mode, with a 100 MHz spectrum slice, using TDD technology, and employing 3 cells to cover the entire 1200 squared metres of space inside the lab [13]. All three cells employ the same frequency in order to improve handover.


(a) 5G Core and Base Station

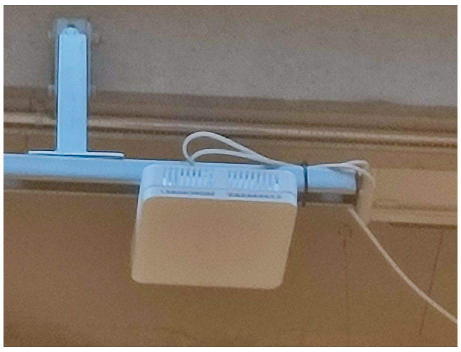
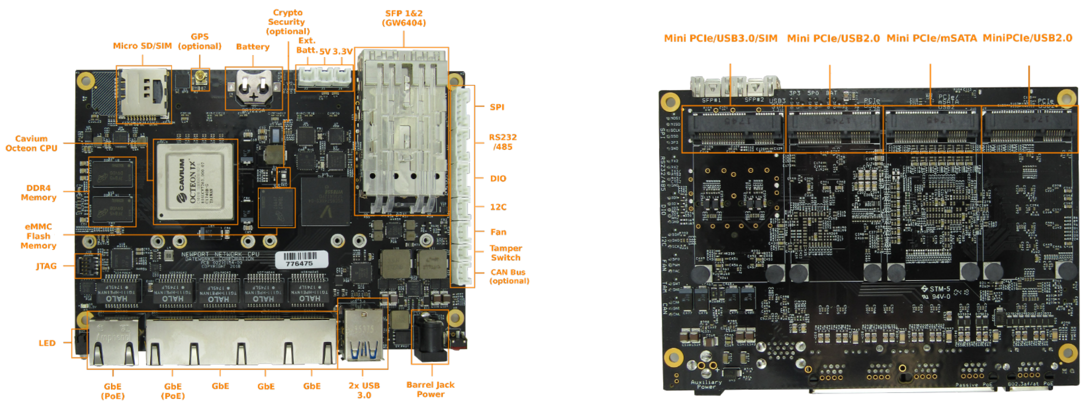

(b) 5G Access Point

Figure 2.6.- Private 5G infrastructure at the AAU 5G Smart Production Lab

2.2.2.- Wireless gateway

In order to give 5G connectivity to devices that do not implement it out of the box (something still very typical nowadays in the industry), a 5G to Ethernet gateway was used. This ”5G box” is composed of a Gateworks Neport GW6404 single board computer (SBC), and a SIM8262E-M2 5G modem. This last component is the modem that gives 5G connection to the device using a SIM card, and the Gateworks SBC acts as a router that redirects all IP/Ethernet traffic from the connected device to the 5G network.

Figure 2.7 depicts the hardware layout of the gateway. The main specs of the computer are a 4 core 1.5GHz ARM SoC processor, 2 GB of DDR4 RAM, 8GB of eMMC Flash Memory, 4 mini-PCIe slots, 5 Ethernet ports and native support for CAN bus and GPS [32]. The mini-PCIe slots are what enable connectivity with the 5G module and many other, like the Intel AX210 modem that is compatible with even the WiFi 6E specification. It comes with a specific distribution of Ubuntu 20.04 that is purposely made by Gateworks for these devices.



(a) Top

(b) Bottom

Figure 2.7.- 5G to Ethernet gateway SBC from the top and bottom

The 5G modem used in them is the Simcom SIM8262E-M2, compatible with 5G, 4G (LTE) and even 3G (HSPA+) networks [33]. Following 3GPP Release 16, it provides connectivity to both 5G Non Stand Alone (NSA) and Stand Alone (SA) networks, and can handle up to 3.4 Gbps of Downlink data transfer and up to 1 Gbps of Uplink. It also supports many different Global Navigation Satellite System technologies, and it can be used in many different ways with PCIe, USB or even GPIO adapters. In this case, it will be connected to the mini-PCIe slots on the Gateworks modem.

Figure 2.8.- SIM8262E-M2 5G modem

2.2.3.- UR robotic arms

Universal Robots (UR) is a Danish manufacturer of collaborative robots, commonly known as cobots, that started out in 2005 and has become one of the largest robotic arm manufacturers in the world [34]. These robotic arms are designed to work alongside humans in a shared workspace, and they are renowned for their versatility, ease of use, and safety features. They are commonly found along different industries for tasks such as assembly, pick and place, welding, and quality inspection.

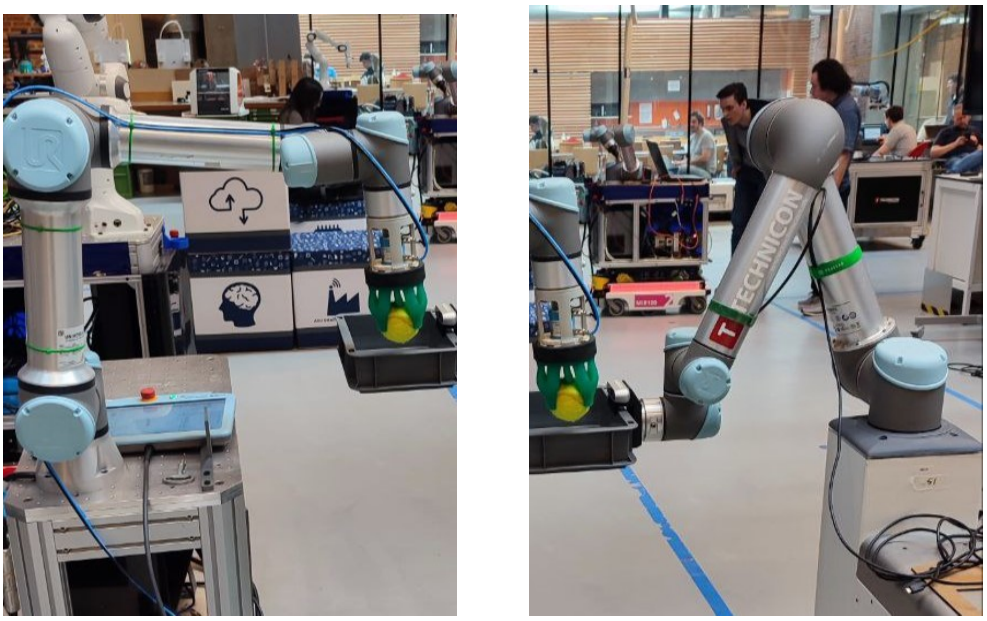

(a) UR5e with gripper (b) UR5 with tray holder

Figure 2.9.- UR5e and UR5 robotic arms

In the experimental work, two very similar robotic arms were used: the UR5 and the UR5e. They both have a payload of 5 kg, a 85 cm reach with the arm, and 6 Degrees of Freedom (DoF). The main difference is in their repeatability, as the UR5 has 0.1 mm of uncertainty while the newest robot has 0.03 mm. In addition, the UR5e also has an integrated force/torque sensor for improved precision and sensitivity, a more user-friendly interface, and even more safety features. But in general, they are pretty much the same robotic arm and they work the same way.

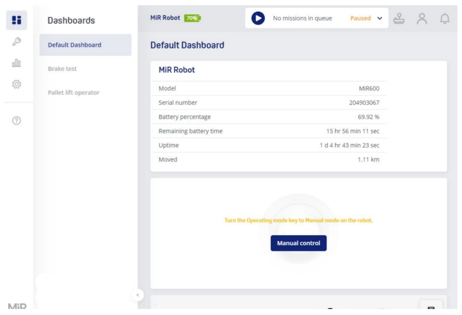
The last difference that the two robotic arms that were present at the laboratory were in the attached gripper. Figure 2.9 illustrates the visual differences. The UR5 arm didn't had a gripper, it just had an adapter to grab trays made with a 3D printer. Meanwhile, the static robot had an air pressure machine next to it that enabled the robot to have a compressed air gripper from The Gripper Company [35]. These grippers are completely made out of washable materials that even allow the robots to handle food, and they are designed for grabbing delicate objects, The used one in specific was the Four Fingers Lip model, with model number TGC-SFG-4X-MRIB-XX.

For controlling both the UR5 and UR5e robotic arms, the Real-Time Data Exchange (RTDE) interface is used. It is a real-time communication protocol on top of port 30004 of TCP [36]. The RTDE allows users to send commands to the robot and receive data directly from its registers, enabling precise control and monitoring of the robot's operations.

2.2.4.- MiR robots

MiR (Mobile Industrial Robots) is a leading manufacturer of Autonomous Mobile Robots (AMRs) designed for industrial and commercial applications. These robots are designed to transport goods and materials autonomously within industrial and warehouse environments. With their advanced navigation capabilities, collaborative operation, and seamless integration with existing systems, these robots help businesses improve productivity, reduce labor costs, and enhance workplace safety.

(a) MiR 100 AMR [37]



No missions in queue

Default Dashboard

MiR Robot

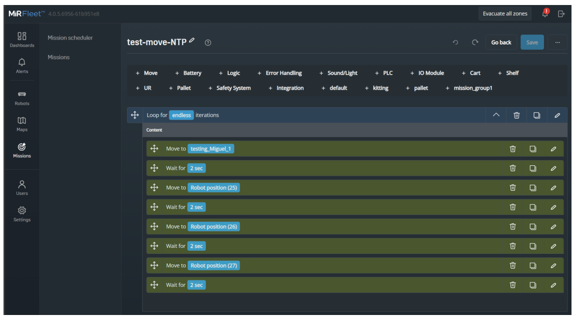
Model	MiR600
Serial number	204903067
Battery percentage	69.92 %
Remaining battery time	15 hr 56 min 11 sec
Uptime	1 d 4 hr 43 min 23 sec
Moved	1.11 km

Turn the Operating mode key to Manual mode on the robot.

(b) MiR v3 dashboard webpage

Figure 2.10.- MiR 100 robot and its graphical control webpage

The MiR robots present at the AAU 5G Smart Production Lab are 100 and 200 models as depicted in Figure 2.10, commonly found in many companies. They both offer around 10 hours of continuous use, a maximum speed of up to 1.5 m/s (only 1 m/s with payload), and a maximum transport payload of 100 kg and 200 kg respectively [38]. These robots are basically composed of two motors with their controller, two SICK safety LIDARs and two Intel RealSense cameras along many other small sensors, various safety equipment, a NUC computer and a WiFi router to give wireless communications [39].

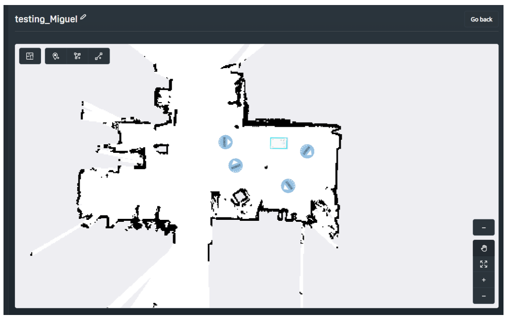


test-move-NTP

Loop for endless iterations

Move to testing_Miguel_1
Wait for 2 sec
Move to Robot position (25)
Wait for 2 sec
Move to Robot position (26)
Wait for 2 sec
Move to Robot position (27)
Wait for 2 sec

(a) MiR Fleet Manager v4 Mission



(b) MiR Fleet Manager v4 Map

Figure 2.11.- MiR 100 robot and its graphical control webpage

The whole control of the robot is intended to be conducted over their webpage interface, which is accessed by putting the robot's IP in a web browser. From that Graphic User Interface (GUI), the user can configure activities, missions (a group of activities), the surrounding map with checkpoints and forbidden zones, and many other, as represented in Figure 2.11.

The biggest difference present between some of AAU's MiR robots is their software version, as some use version 3 and others version 4. This translates into the ones on v3 being run independently (the robot directly takes the assigned tasks/missions), while v4 robots rely on a centralised Fleet Manager that sends the assigned missions to the robots, so the user has to program their behaviours from the MiR Fleet Manager webpage. For this thesis project, MiR v3 software is used, as the lent robot was running that version, and it actually makes the 5G integration easier.

2.2.5.- ER robot

Enabled Robotics is an integrator company of mobile robots and robotic arms that has been operating since 2016 [40]. Not only they combine both into a unique product, but they also develop specialized software for them to make its programming and integration with other software much easier. As it is depicted in Figure 2.12, they use a block programming interface in the robot's webpage to easily configure the desired tasks for the robot.

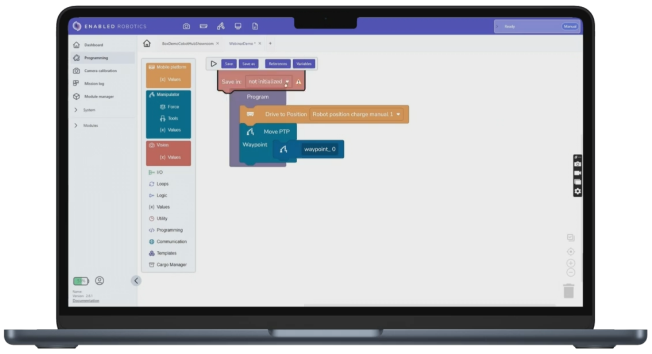


Figure 2.12.- Block programming of the ER robots [41]

The ER robot that the university owns is a MiR 200 model mobile platform that has the previously mentioned UR5 robotic arm nicely integrated on top. At the tip of the arm there is not a gripper, but a 3D printed tray holder that also integrates an Intel RealSense D435 camera with depth sensing capabilities. All these components integrate into a package that, combined with the block programming on the webpage, allows to easily configure the robot to make tasks like calibrating the arm's position to pick up a tray by the use of checkerboard markers, as shown in Figure 2.13. This allows for very fast configuration of simple tasks, making it very easy to integrate these robots into a factory. But it also comes with drawbacks, as the options for communicating with other devices are quite limited, making it impossible to integrate it over 5G. That is why the the robot was used for the project, but not the actual software that came with it.

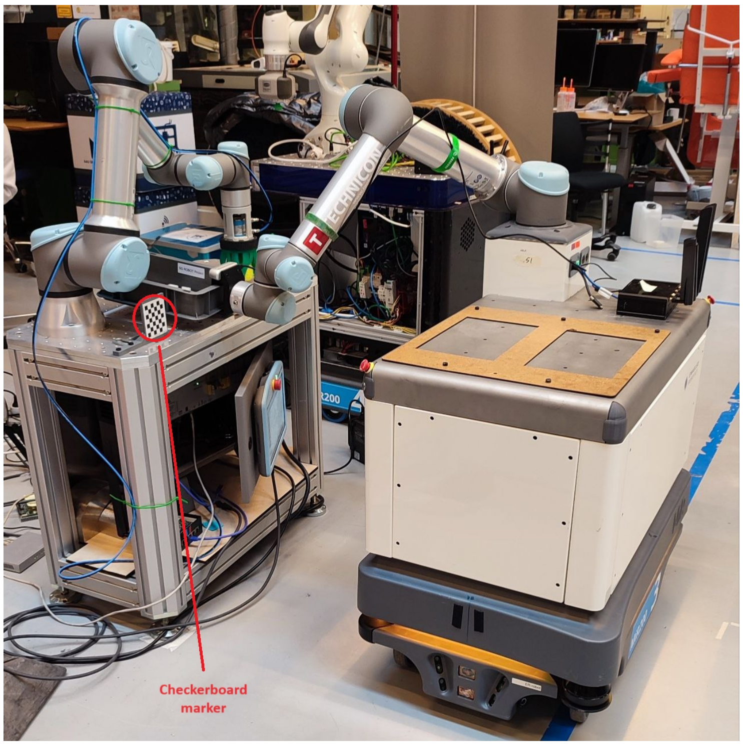


Figure 2.13.- ER robot picking up a tray using a checkerboard marker as positional reference

3. Time Synchronization in industrial 5G systems

This chapter sets the basis of the practical work done in this thesis by presenting an initial study of multi-robot time/clock synchronization in industrial settings using wireless 5G technology.

3.1.- Introduction

There are many reasons on why computers need to have synchronized clocks between devices [42]. The main one is for data synchronization in networking and communications. You could think that it is important for multimedia applications such as live video conferencing or streaming, but the most important one is probably for money transactions like in the stock market, or between banks. It is also important in cybersecurity to protect from DDoS attacks for example [43]. In this industrial case specifically, it is needed that all components in the network have very accurate synchronization because the different robots need to move in coordination with very precise timings. But how is it done?

Every computer has a Real-Time Clock (RTC) chip powered by a small battery that keeps the current date and time even when it is turned off. But this method is not very precise, as every hour the clock may drift up to a few seconds a day. If you want to get the absolute best accuracy, you would either need an atomic clock (very difficult and expensive to get), access to a GPS signal, which is expensive and tricky to get (specially inside buildings)[44], or rubidium clocks [45], which are also very expensive for consumer electronics. That is why clock synchronization in distributed real-time systems has been a big research topic for many years [46] [47], with protocols such as Network Time Protocol (NTP) or Precision Time Protocol (PTP) being the most common ways to reach accurate synchronization in networked systems nowadays [48].

3.2.- Network Time Protocol

The Network Time Protocol (NTP) is a protocol designed to synchronize the clocks of the computers to Coordinated Universal Time (UTC) [49]. This is a global timescale independent of geographic position, it is used by national laboratories and disseminated by radio, satellite and telephone modem. It is the most commonly found option to synchronize devices over the Internet and private networks. This is because it is very simple while offering good precision, it has a wide support, and it doesn't require specialized hardware. In addition, with some calibration, its results can significantly improve [50].

As displayed in Figure 3.1, NTP uses a hierarchical system of time sources to synchronize, each divided into stratum levels. The first one, Stratum 0, are the ultra highly accurate time sources, such as atomic or GPS clocks. These sources are then connected to the primary servers to spread the time reference, and are considered Stratum 1. That is why, when you create a NTP server, the best Stratum you can get if you do not have an available time source it will be Stratum 2, as it will be connected to one of the reference servers. And more than likely, the common devices used in a network are part of the Stratum 3 or below.

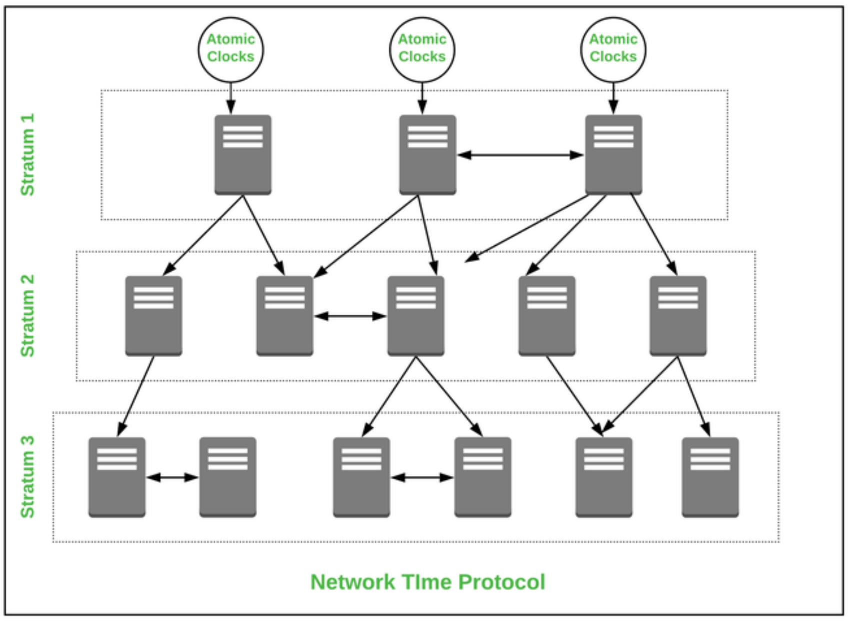


Figure 3.1.- NTP Stratum architecture [51]

Even though the higher the Stratum, the more accurate and reliable the reference, in this testing a local server connected to Stratum 1 references was used. That is why the NTP server in the network is Stratum 2, and the connected devices would be Stratum 3.

The basic synchronization algorithm for NTP is very simple, as it only requires two network packets with a total of four timestamps, and doesn't need the server to store any client information. The origin timestamp T1 is recorded upon departure of the client request. Next, the receive timestamp T2 is registered when the packet arrives at the server, and then the transmit timestamp T3 is reported just before the departure of the server reply.Finally, the destination timestamp T4 is recorded once the message arrives to the client. With these timestamps, both the clock offset and round trip delay samples can be calculated using the following formulas:

$$Offset = [(T2 - T1) + (T3 - T4)]/2$$

$$RTT = (T4 - T1) - (T3 - T2)$$

3.2.1.- Chrony

Chrony is a modern implementation of the Network Time Protocol (NTP) used for synchronizing computer clocks over a network. It is the reference NTP client and server in practical industrial implementations, as it is more accurate, robust and offers more tools than its predecessor (*ntpd*). It is designed to provide accurate timekeeping while being robust and resistant to network disturbances and fluctuations. It performs well in a wide range of conditions, including intermittent network connections, heavily congested networks, changing temperatures (ordinary computer clocks are sensitive to temperature), and systems that do not run continuously, or run on a virtual machine.

3.3.- Robust NTP synchronization in industrial 5G

With the aim of understanding the reference level of synchronization with simple NTP solutions over industrial 5G, an initial experiment was proposed. The experiment was oriented to investigate and characterize the accuracy of 5G wireless synchronization in both static and mobile industrial conditions.

The network architecture used for the time synchronization testing over 5G is depicted in Figure 3.2. The 5G network is very simplified, but the important concept to know is that the Chrony NTP server is running in the main router of the private network (192.168.100.1 IP). This router is directly connected through cable to the whole private 5G network inside the laboratory, which gives wireless connection to the 5G gateway running the Chrony NTP client. This gateway is on top of a mobile MiR robot in order to asses both the static and mobile cases.

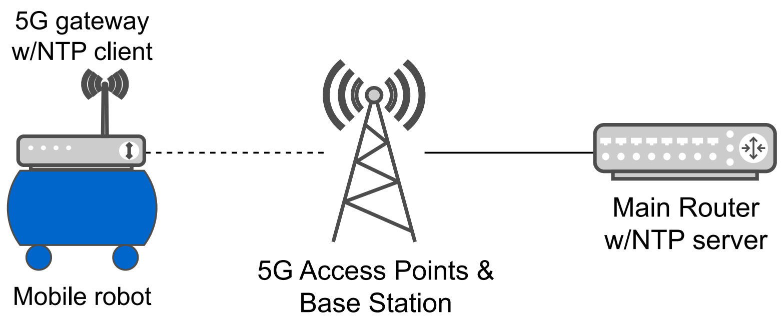


Figure 3.2.- NTP architecture for 5G synchronization testing

The typical achieved accuracy between two machines synchronised over the Internet is within a few milliseconds, and the accuracy in wired LANs is typically in tens of microseconds [52]. With wired links and hardware time stamping, or a hardware reference clock, sub-microsecond accuracy may be possible [53]. However, this is not the aimed scenario, as most of the industrial devices used do not have those capabilities, and wireless networks are more unreliable in their synchronization.

As explained in subsection 2.1.1 for URLLC use cases over 5G, the desired latency should be around the 1 ms mark, so consequently, the synchronization level has to be better, even with very simple time synchronization solutions. The related Key Performance Indicator (KPI) for this section is to achieve synchronization levels one order of magnitude lower, around the 0.1 millisecond mark. Even though more advanced and complex solutions exist that offer better metrics [54], this solution could pose more usage of hardware and network resources. The target in this thesis is to use very simple synchronization mechanisms, such as a NTP implementation with Chrony, to achieve adequate results for typical industrial scenarios.

3.3.1.- NTP server configuration

The Chrony configuration of the NTP server was kept quite simple, as it was the default one. That way, variables were minimized, and as it is a router directly connected to the Internet Service Provider (ISP), the time reference that it gets from the NTP pool is very reliable. In addition, it was not desirable to change anything on the main router, in order to not affect in any moment the rest of the network.

The actual Chrony configuration file is listed next. As Ubuntu 20.04 was used, the file is located at "/etc/chrony/chrony.conf". First, the NTP server is specified, which is the default NTP pool of servers, along with specific modes for updating the client faster in startup The minimum and maximum polling intervals were kept at default values, which are kept at 10 (every 1024 seconds, as the number refers how many times 2 is multiplied to get the seconds). Next, they are specified the path to save the keys for NTP authentication; the path to the file that stores the estimated rate of the clock's drift, in order to increase accuracy while it's not synchronized; the NTS dump directory to save keys and cookies; and the log directory of Chrony. The "maxupdateskew" directive sets the threshold for determining whether an estimate might be so unreliable that it should not be used. Afterwards, the "makestep" directive allows to quickly correct the system clock if the offset is bigger than 1 second for up to 3 times after Chrony starts, in this configuration. Then, "rtcsync" is related to enabling the synchronization of the system clock with the hardware Real-Time Clock (RTC), helping maintaining the correct time across reboots, and "leapsectz" specifies the time zone file to use for leap seconds. Finally, the last two directives allow clients from those networks to connect to the server, serving the NTP time reference. These networks are just the internal wired and WiFi network inside the lab (192.168.100.0/24), and the private 5G network that is also present (10.94.0.0/24).

```
pool pool.ntp.org iburst
keyfile /etc/chrony/chrony.keys
driftfile /var/lib/chrony/chrony.drift
ntsdumpdir /var/lib/chrony
logdir /var/log/chrony
maxupdateskew 100.0
```

```
makestep 1 3
rtcsync
leapsectz right/UTC
allow 192.168.100.0/24
allow 10.94.0.0/24
```

3.3.2.- Base NTP client configuration

The base Chrony configuration used inside the configuration file, found at ”/etc/chrony/chrony.conf” in Ubuntu 20.04, was the following one:

```
server 192.168.100.1 minpoll -6 maxpoll 2 iburst
keyfile /etc/chrony/chrony.keys
driftfile /var/lib/chrony/chrony.drift
ntsdumpdir /var/lib/chrony
logdir /var/log/chrony
maxupdateskew 100.0
makestep 1 3
rtcsync
leapsectz right/UTC
```

This is a very typical configuration for a Chrony client. First of all, the NTP server is specified, as well as the minimum and maximum polling intervals, along with specific modes for updating the client. This translates into a typical polling rate of 4 seconds (determined by the ”maxpoll” directive), but with opportunities to update faster the clock, specially at startup. Next, they are specified the path to save the keys for NTP authentication; the path to the file that stores the estimated rate of the clock’s drift, in order to increase accuracy while it’s not synchronized; the NTS dump directory to save keys and cookies; and the log directory of Chrony. The ”maxupdateskew” directive sets the threshold for determining whether an estimate might be so unreliable that it should not be used. Afterwards, the ”makestep” directive allows to quickly correct the system clock if the offset is bigger than 1 second for up to 3 times after Chrony starts, in this configuration. And the last ones are related to enabling the synchronization of the system clock with the hardware Real-Time Clock (RTC), helping maintaining the correct time across reboots, and finally, specifying the time zone file to use for leap seconds.

The main change with respect to the default one was that the NTP server was changed from a public internet one to the private one running on the router at ”192.168.100.1” IP address, and the use of the ”minpoll”, ”maxpoll” and ”iburst” directives. These were specified to better ensure a reference configuration from which to compare changes made afterwards.

3.3.3.- Base NTP client measurements

Once both the NTP server and client were configured, the time synchronization measurements over 5G could be taken. This was done recording every second the ”Last Offset” parameter that the command ”chronyc -n tracking”. Three different runs of more than two hours were done for the static case, and then another three for the mobile case. For the static case, the mobile robot was just left turned on without moving, while for the mobile case, the robot was moving across the laboratory at a maximum pace of 1 m/s. The route was specifically designed to make the robot change between two of the 5G Access Points in the laboratory, located one at each end.

All the six resulting runs are represented in Figure 3.3. Considering the defined 0.1 ms target in the KPI, the results were not satisfactory. While in general, the offset is low and contained below 0.1 ms, at certain instances spikes of up to +/-2.5 ms are experienced. It was identified that this problem is something that could be alleviated with a better configuration of the client, as most of these peaks were in both directions (positive and negative offset), suggesting that they could be due to bad estimates that then have to be corrected. Nonetheless, the performance for both static and mobile cases is very similar, which indicates that the NTP synchronization is robust to dynamic multi-path radio propagation and handover scenarios, independently of the movement of the 5G terminal (at least for the considered use cases).

3.3.4.- Optimized NTP client configuration

Once observed the undesired behaviour in the clock synchronization, the configuration was tweaked to better tailor the 5G SA network in the AAU 5G Smart Production Lab. The first

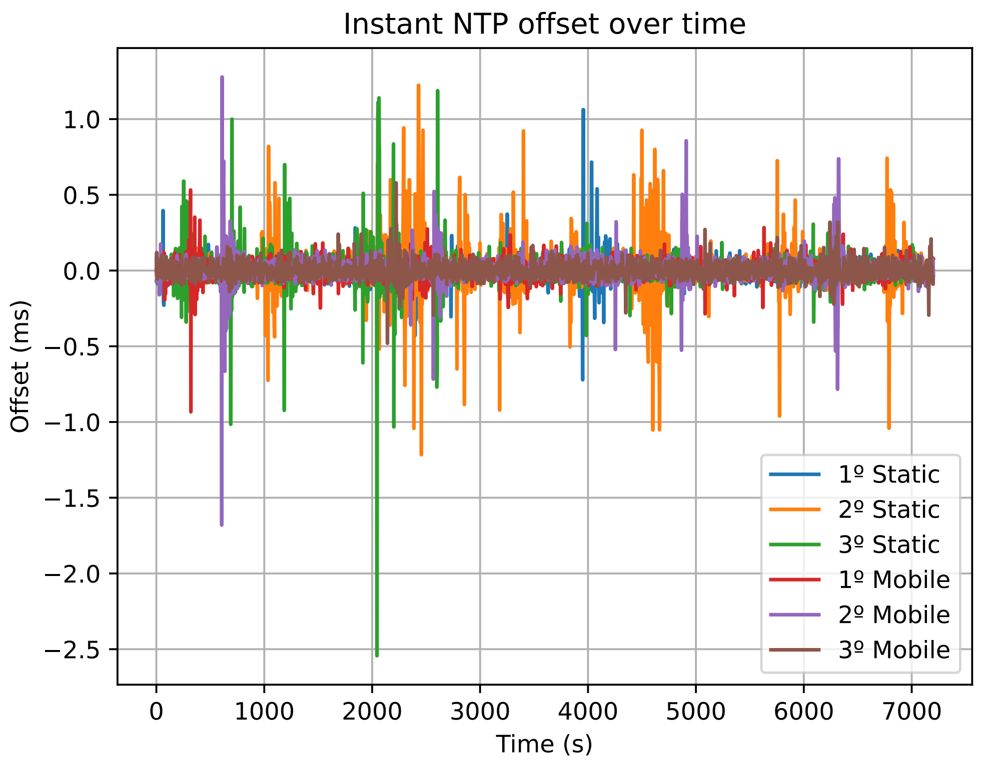


Figure 3.3.- Baseline 5G NTP synchronization accuracy for the different reference cases

change made was to add the ”maxdelaydevratio” directive, as it discards NTP packets that take too long to arrive in comparison to the mean delay. Setting it at a deviation ratio of 2.6 seemed to be the best, as below that there could be too many packet discards, and above that there would be unreliable estimates. This value depends on network congestion and the performance of the network, so it could be different for other environments.

Next, the ”xleave” directive was added, as it improves accuracy by enabling the server to respond with more accurate timestamps on transmission. Included as well was ”maxslewrate”, because it limits how quickly Chrony can adjust the clock, balancing between rapid corrections and avoiding abrupt changes that could affect system stability.

In addition, both the ”maxupdateskew” and the ”makestep” directives were modified. The first one was reduced to just 3 parts-per-million, to help reduce the effect of unreliable estimates.

Meanwhile, the other one was modified to help correct faster the clock if the program has just started.

```
confdir /etc/chrony/conf.d
server 192.168.100.1 minpoll 0 maxpoll 1 iburst xleave maxdelaydevratio 2.6
keyfile /etc/chrony/chrony.keys
driftfile /var/lib/chrony/chrony.drift
ntsdumpdir /var/lib/chrony
logdir /var/log/chrony
maxslewrate 30
maxupdateskew 3.0
makestep 0.5 20
rtcsync
leapsectz right/UTC
```

3.3.5.- Optimized NTP client measurements

Once the NTP client had an updated configuration, the time synchronization measurements over 5G could be taken. This was done recording every second the ”Last Offset” parameter that the command ”chronyc -n tracking”, just as described in Subsection 3.3.3. The mobile case was set up the exact same way as in the previous measurements, but in this round of measurements, only one run for each case was possible.

Figure 3.4 depicts the instant offset that the Chrony client in the 5G gateway calculates through NTP with respect to the server, which is the main router in the laboratory, before and after the configuration change. First of all, the synchronization after it is mostly below the 0.1ms, something that was impossible with the baseline configuration. But most importantly, the big spikes that regularly throw off the synchronization more than +/-1 ms are now gone. This improvements translates into a very reliable clock synchronization between different devices in the 5G network, meeting the previously set mark for the time synchronization KPI below +/-0.1 ms (at least for more than 99% of the time).

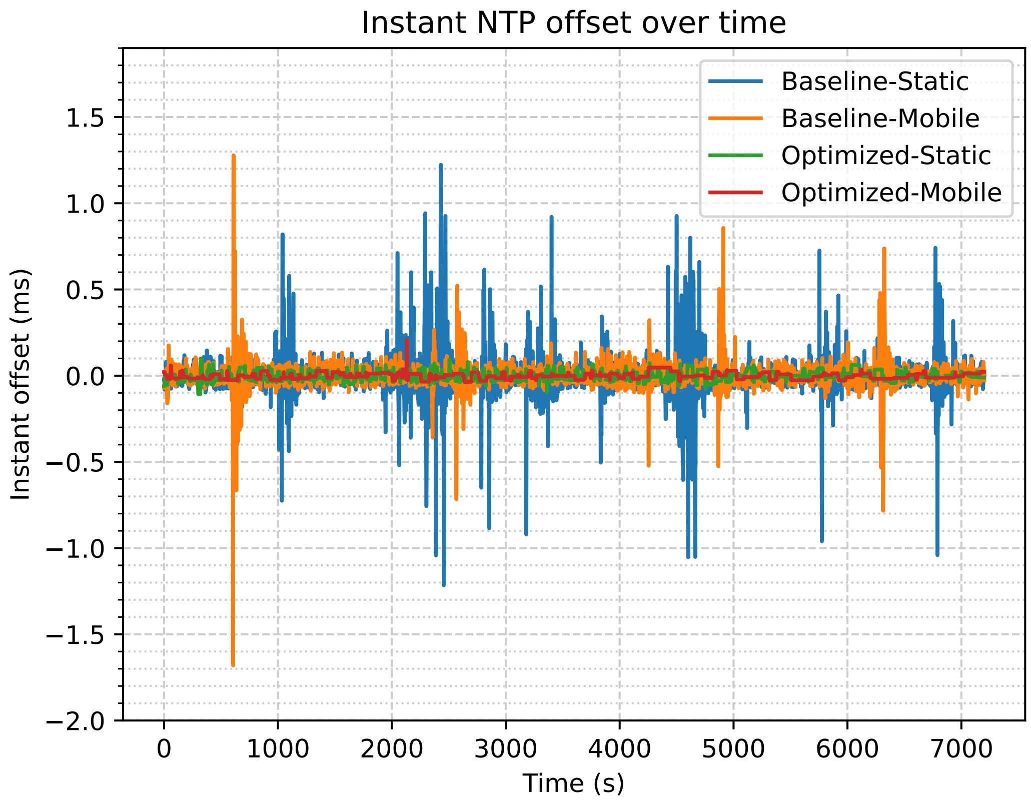


Figure 3.4.- Optimized 5G NTP synchronization results as compared to the baseline ones

In Figure 3.5, a Cumulative Distribution Function (CDF) plot of the baseline measurements and the new ones with the optimized configuration are shown. It is worth noting that in this representation, the ideal representation would be a line that is vertical in the 0 ms offset mark, so the closer a representation is to the centre, the better. Even though the crossover between offsets is not completely centered at the 50% mark, specially in the case after the modifications, that is not important, as it could just be due to an internal RTC clock that is going a bit faster or slower than Chrony had previously calculated. The crucial aspect shown in this graph is that the offset distribution is much tighter, specially in the upper bounds of the offset.

The offset after the configuration changes is nearly 10 times better than before in the 1% and 99% percentiles. Only in the peak cases it goes out of the +/-0.1ms bound, but those are only on the worst 0.1% of the time. Meanwhile, with the baseline configuration it could go even

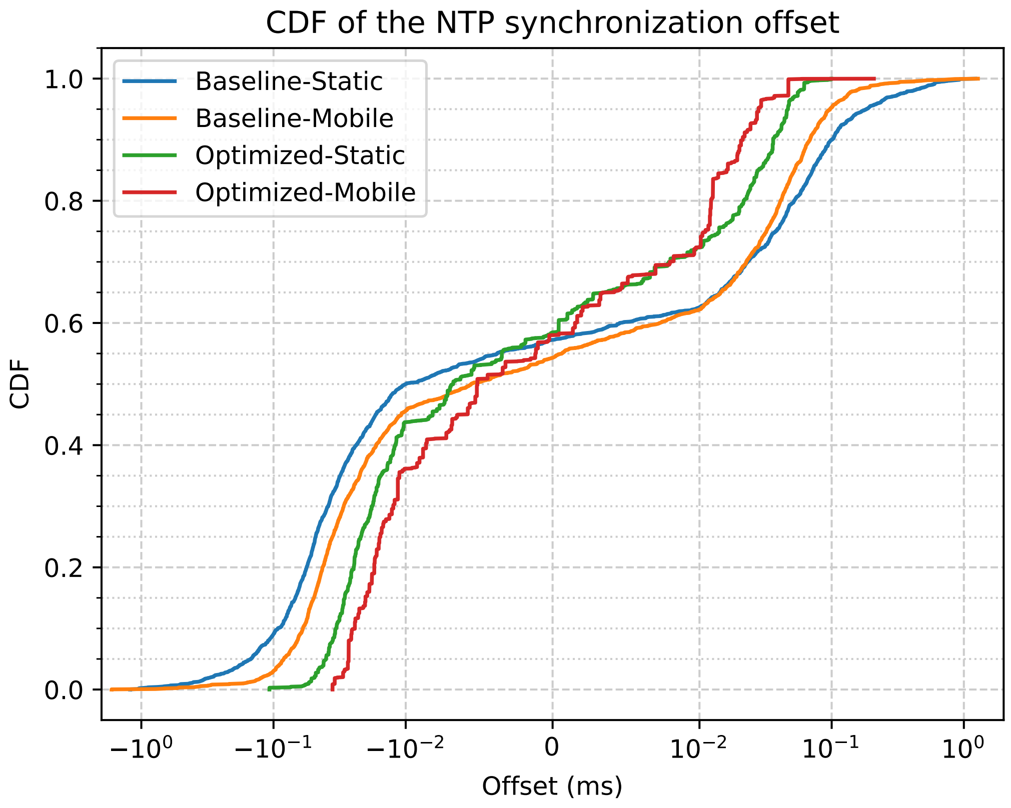


Figure 3.5.- Statistical representation of the optimized 5G NTP syncrhonization results as compared to the baseline ones

out of the +/-1ms bound. Another important remark is that in both the baseline and optimized configurations, static and mobile cases do not show a significant difference.

3.4.- Time synchronization conclusions

To conclude this chapter, it is necessary to make a recapitulation of the objectives and KPIs set. From all the objectives, the completed work fulfills the first objective ”OBJ1”: investigation and characterization of time synchronization in 5G networks. This investigation was carried out using Chrony, an NTP implementation that was configured on the Main router as a server and on the 5G gateway as its client. Both static and mobile industrial scenarios were considered for two different client configurations, although it was observed that this did not translate into any meaningful difference in the results. The

baseline configuration did not yield the desired outcome set for the KPI beforehand as occasional peaks threw the time synchronization by more than +/-1 ms. Nevertheless, the optimized one did improve the previous results by managing to operate 99.9% of the time below the +/-0.1 ms offset threshold. As a result, the initially set KPI threshold for this experiment over 5G has been met with the optimized configuration of the Chrony client, meeting the industrial reliability specification.

4. Design and Development of an Industrial Testbed with Coordinated Robots

This chapter builds up on time-synchronized 5G-based multi-robot systems and proposes a real-world use case implementation taking advantage of the learnings from Chapter 3.

4.1.- General requirements

Different aspects need to be considered before the design of an industrial testbed demonstration with coordinated robots. There are many possible use cases in the industry, ranging from time-critical, relaxed real-time, and delay-tolerant. As a reference, these 3 use cases categories (UCCs) [55] are described from a manufacturing process perspective:

- UCC1 ”Time critical processes”: they include real-time closed-loop robotic control, video-driven machine-human interactions, and Augmented Reality/Virtual Reality for maintenance and training. They all have a direct impact in manufacturing efficiency, yields and safety.
- UCC2 ”Non real-time processes inside the factory”: they comprise tracking products and machine inventory, non-real-time sensor data, and remote inspection and diagnostics. They all support the management at the production facilities.
- UCC3 ”Enterprise communication”: it comprises the warehousing and logistics planes, with employee and back-office communications and tracking of post-production goods.

Obviously, the most difficult use cases to implement over wireless networks are the ones in UCC1, as they are the most dependant on the network performance to succeed and ensure the safety of the workers in the factory. As it was previously discussed in the introduction, the expected use case that the demonstration is required to cover is the ”Cloud Control of Machines”. This will enable to rely on virtualized computing resources, storage, applications, and services that are managed by software so the data generated can be accessed on-demand, instead of being dependant on programmable logic controllers that are hardwired into computing systems to implement the control management.

Prior to the use case definition, a baseline use case was implemented, where two static arms passed a tennis ball between each other using the same concept of a 5G connected cloud controller. Even though this already requires good synchronization between the computers, it lacks the complexity of coordinating a mobile platform in addition to the movements of the arms. That is why, as exposed in Chapter 3, the first thing that was needed was to ensure precise time synchronization between the 5G gateways that are going to be used.

After discovering the synchronization limitations over 5G, learning about the different types of industrial use cases, and being trained on how to operate the robots, the next step was to decide the actual application to implement. Many meetings and talks were held with various stakeholders, mainly inside the Wireless Communication Networks and the Manufacturing and Production departments at Aalborg University, in order to get directives and recommendations on what was relevant, novel, and technically-feasible to be considered. Figure 4.1 contains the sketch representing the proposed industrial use case demonstration idea.

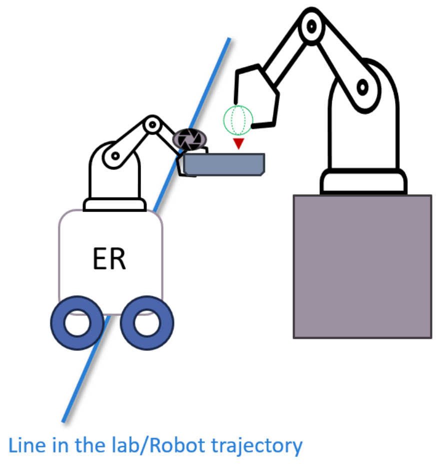


Figure 4.1.- Sketch of the proposed use case idea

The main idea, depicted in Figure 4.1, was to pass an object, in this case a tennis ball, from the static arm to the tray of the mobile robot without stopping, as that is something that requires coordination and synchronization. This idea was well received by all stakeholders, as it meant demonstrating an industrial use case scenario that is not seen in current factories, and that could serve to promote more flexible and efficient use cases with robots, as traditionally the robot would need to stop. The use case was inspired in [56], where 4G LTE was used, leading to a not very reliable performance.

There are a few steps needed to implement the desired use case:

1. Two time synchronized 5G gateways, one for the static UR robotic arm and another for the mobile ER robot, to execute the Slave programs, and a Virtual Machine in the Edge-Cloud to execute the Master program that will control them. The full physical architecture will be outline in Section 4.2, the next one.
2. The static robotic arm needs to have a gripper and an object (for example, a tennis ball), and the mobile robotic arm needs something to grab the object (for example, a tray), so that it is possible to pass the object from one arm to the other.
3. The mobile robot will start from a specific position, and it will move in a straight line at a constant pace towards the static robot.
4. Once that the mobile robot enters a specified zone at which the exchange can be made, without stopping the movement, the static robot will drop the tennis ball into the tray, and both robots will resume normal operation afterwards.

It was a big concern to know whether the object exchange without stopping was possible, as the tray was only 14 cm wide. Is a 14 cm margin enough to make the use case work? Operating at a speed of 0.5 m/s (typical for industrial robots), and knowing that the available distance is 0.14 m, that leaves a total of 280 ms for the ball to drop into the tray. That could seem like a lot, but taking into account that it drops from 12 cm, the ball already takes around 160 ms to drop. In addition, both the position request and the gripper activation have to be taken into account, as the position request usually takes 30 ms and the gripper activation with a typical air solenoid valve can take up to 50 ms [57]. That leaves a maximum total of 30 ms to send the message from the mobile robot to the static robot without dropping the ball.

4.2.- Physical architecture

The physical architecture of the devices that interact in the demonstration is depicted in Figure 4.2.

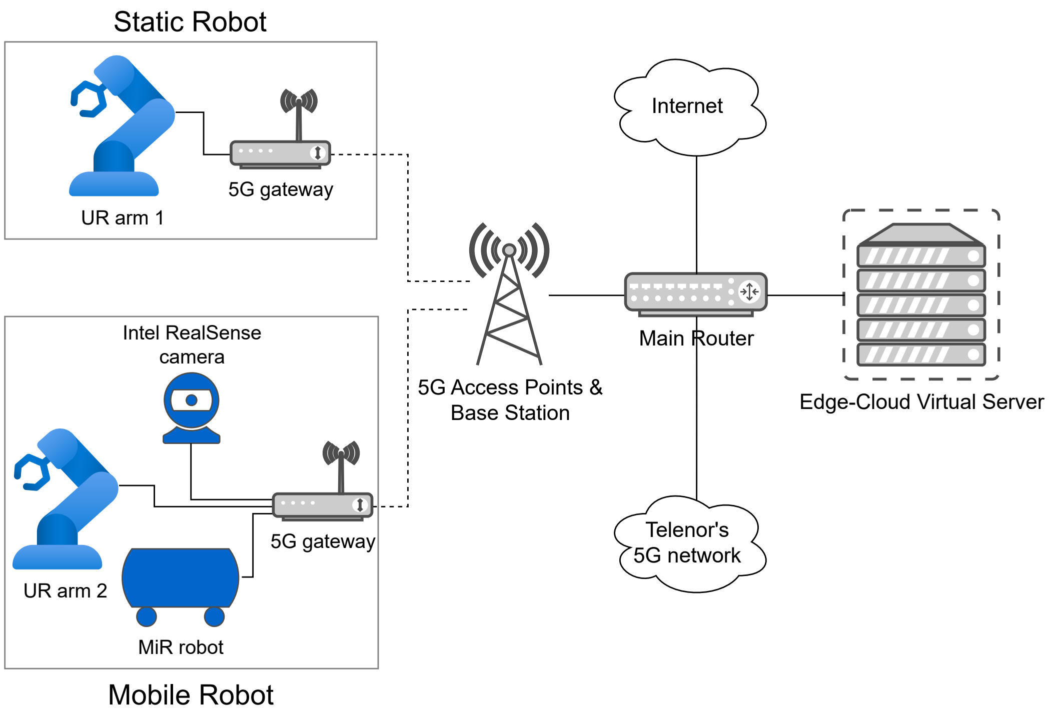


Figure 4.2.- Demo physical architecture

On the left, both the static and the mobile robots are represented. The static one involves the UR5e robotic arm, with its internal computer directly connected to the Gateworks gateway in order to provide 5G connectivity, hold the NTP client for the time synchronization, and run the slave program for the demo. The mobile robot is a bit more complex, as everything is built on top of the MiR robot platform. Both its UR5 arm and the 5G box are connected to its internal switch, creating a cabled network. In addition, the iRS camera is connected to the gateway via USB, as some testing was done to stream its live content to the server, although in the end it was not used.

Both 5G routers are connected to the 5G private network using their integrated modems and SIM cards. This network is composed of a few radio access points along the AAU 5G Smart Production Lab, and a private 5G core that is connected to both the Internet and the Telenor 5G network (a Danish telecommunications company) using a general router. The on-site Edge-Cloud servers are also directly connected by cable to this router, in order to run virtual instances like the one used for the Virtual PLC. The main router is also running the NTP server used by all devices to synchronize time.

4.3.- Software architecture

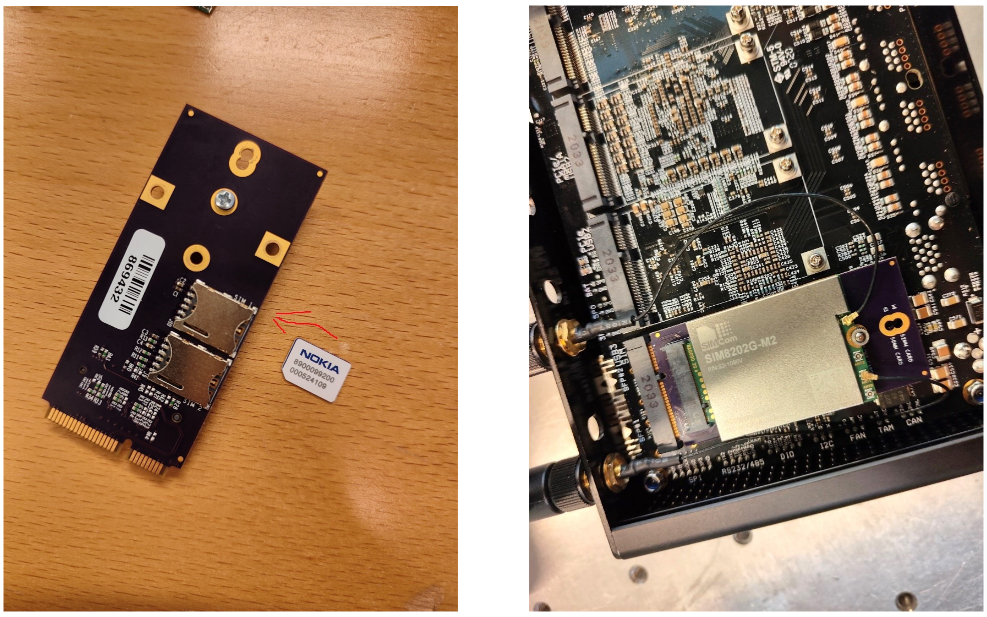
In order to implement the use case coordination and functionality, a number of SW solutions were developed, and implemented over the described physical architecture. The baseline NTP-based over 5G explained in Chapter 2 was implemented, and MQTT was leveraged for control message exchange between elements for functional operation.

4.3.1.- 5G gateway configuration for IT/OT integration

This subsection explains the required hardware and software configuration for one of the two 5G gateways that is depicted in Figure 4.2. As two gateways are required, this would mean repeating the process two times, just ensuring to adapt it for the different IPs that are used in each case.

The first step is to add the 5G modem to the Gateworks gateway if it is not included already. This is done by getting a mini PCIe daughter board and a private NOKIA SIM card that had already been activated for the 5G network. Both are combined and then inserted into the gateway's motherboard as shown in Figure 4.3 (a). Then, the selected 5G modem is inserted into the daughter PCB with just one screw, and the antennas can be connected to the module as displayed in Figure 4.3 (b). Be aware that, if there are not enough antennas to connect to all the available connectors on the 5G modem, it is needed to make sure that they are connected to the ports used for the N78 band, as the standalone 5G network runs on it.

Once the hardware part is ready, it is time to set up the software. A computer is needed to configure the router using a direct Ethernet connection to the ”blue port” (eth2 interface), as demonstrated in Figure 4.4.



(a) SIM card and daughter board (b) Daughter board and 5G modem into the 5G gateway

Figure 4.3.- SIM card and 5G modem connection in one of the 5G gateways

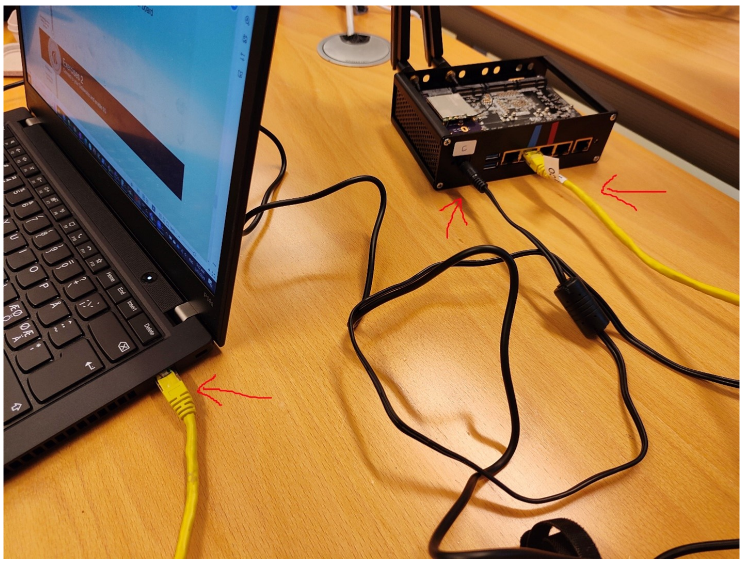


Figure 4.4.- Laptop connected to the configuration port of the 5G router

This network interface should be running a DHCP server that will give the PC an IP in the 10.42.0.0/24, making it possible to connect through SSH to the SBC using the IP 10.42.0.1, as username ”aau”, and as password ”aau” as well.

Once connected, it is needed to create a 5G connection profile in Network Manager with the following command ”sudo nmcli c add type gsm ifname cdc-wdm0 con-name nsa-aau5g apn internet”, and then activate it with ”sudo nmcli c up nsa-aau5g”.

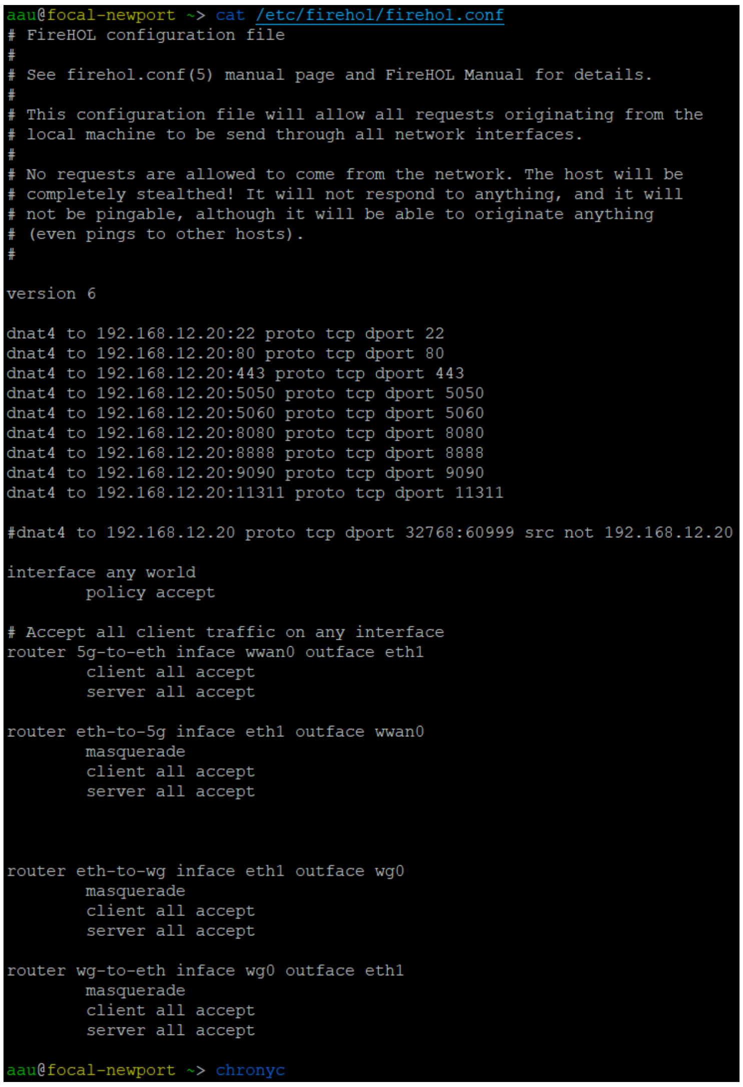
The next step is to configure the firewall to perform routing with NAT and masquerade. These lines were inserted in the FireHOL configuration file at the ”/etc/firehol/firehol.conf” directory, resulting in the image shown in Figure 4.5:

```
dnat4 to 192.168.12.20:80 proto tcp dport 80
dnat4 to 192.168.12.20:443 proto tcp dport 443
dnat4 to 192.168.12.20:8080 proto tcp dport 8080
dnat4 to 192.168.12.20:9090 proto tcp dport 9090

# ----- Content previously in the file -----

router eth-to-5g inface eth1 outface wwan0
        masquerade
        client all accept
        server all accept

router 5g-to-eth inface wwan0 outface eth1
        masquerade
        client all accept
        server all accept
```


```
aau@focal-newport ~> cat /etc/firehol/firehol.conf
# FireHOL configuration file
#
# See firehol.conf(5) manual page and FireHOL Manual for details.
#
# This configuration file will allow all requests originating from the
# local machine to be send through all network interfaces.
#
# No requests are allowed to come from the network. The host will be
# completely stealthed! It will not respond to anything, and it will
# not be pingable, although it will be able to originate anything
# (even pings to other hosts).
#

version 6

dnat4 to 192.168.12.20:22 proto tcp dport 22
dnat4 to 192.168.12.20:80 proto tcp dport 80
dnat4 to 192.168.12.20:443 proto tcp dport 443
dnat4 to 192.168.12.20:5050 proto tcp dport 5050
dnat4 to 192.168.12.20:5060 proto tcp dport 5060
dnat4 to 192.168.12.20:8080 proto tcp dport 8080
dnat4 to 192.168.12.20:8888 proto tcp dport 8888
dnat4 to 192.168.12.20:9090 proto tcp dport 9090
dnat4 to 192.168.12.20:11311 proto tcp dport 11311

#dnat4 to 192.168.12.20 proto tcp dport 32768:60999 src not 192.168.12.20

interface any world
        policy accept

# Accept all client traffic on any interface
router 5g-to-eth inface wwan0 outface eth1
        client all accept
        server all accept

router eth-to-5g inface eth1 outface wwan0
        masquerade
        client all accept
        server all accept



router eth-to-wg inface eth1 outface wg0
        masquerade
        client all accept
        server all accept

router wg-to-eth inface wg0 outface eth1
        masquerade
        client all accept
        server all accept

aau@focal-newport ~> chronyc
```

Figure 4.5.- FireHOL configuration

Once the file is saved, and the firewall is restarted with ”sudo firehol try”, then the gateway should be ready to be connected, in this case to the mobile robot using the red port (eth1 interface), as the interface was configured with the 192.168.12.20 internal IP of the MiR robot.

4.3.2.- Static UR robot

To ease the understanding of the SW implementation at the static UR robot side, Figure 4.6 represents the software architecture and network communications of the program that runs in the 5G box. It is directly connected by cable to the UR robotic arm controller, which has the 10.42.0.233 IP, in order to communicate with the RTDE API running in it at the 30004 TCP port.

For that network interface, the device uses the IP 10.42.0.1, and acts as a DHCP server for that closed network, and as a gateway to the 5G network.

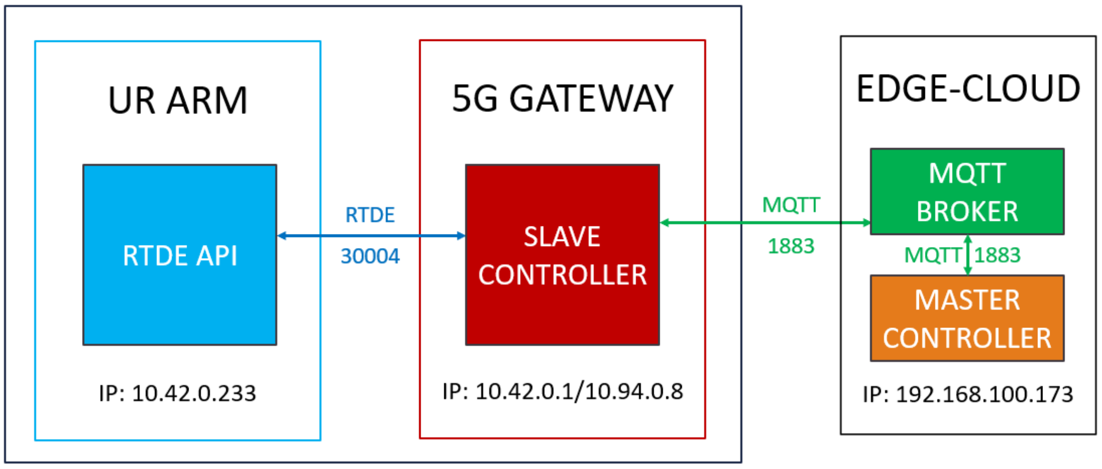


Figure 4.6.- UR robot software architecture

On the other side, the computer is connected to the 5G network in the factory with the 10.94.132.9 IP using the previously described 5G modem and SIM card. The developed slave controller Python program uses MQTT over the 1883 TCP port to communicate with the broker running in the Edge-Cloud server described in 4.3.5. That server also hosts the master controller program, and is connected to the MQTT broker as well in order to communicate with both slave controllers.

Now that the general overview of the software is done, it is time to explain the developed code for the slave controller running in the 5G gateway of the static UR robot. The Python program is called ”mqtt_Slave_static.py”, and it is included in Annex A, at Section A.1.

First, the imported libraries are displayed. Most of them are general libraries very commonly used in Python, like ”sys” in order to access system tools, ”threading” to create different execution threads inside Python, ”time” to get information about the system’s clock, ”socket” to manage network sockets, ”logging” and ”json” to access and save log and json format information, and the ”namedtuple” from collections for a unified way of codification of tuples. In order to manage MQTT messages, the Paho MQTT library was used, and to govern the RTDE communications with the robotic arm, a specific library from UR was used. The last two libraries are internally developed libraries at the university. ”consts” contains general values for both the MQTT and

RTDE communications, like the information to connect to the broker, some general topics used, the RTDE communication rate, or the threshold from which it can be considered that the robotic arm has reached the desired position. Meanwhile, ”my_tools” just contains a few functions to publish MQTT messages, determine if two positions (the actual and the desired) are within the given threshold, and manage csv files to log information. This last library does not follow the best practices when defining its functions, so with Python versions different than 3.9.X, it can give runtime errors.

The beginning of the actual Python code, there are some variable definitions that set the initial position of the robotic arm when launching the code, the UR arm IP and RTDE port, and then the name of the configuration file for the RTDE recipes.

Afterwards, the program will get the recipes from the configuration file and they will be sent to the UR arm. These recipes define the registers the code will access and modify from the robotic arm’s computer. Then, the registers are set up and the joint values for the arm are initialized to the initial values previously defined, and the whole configuration up to this point is sent. Next, the SDO (controlling the gripper) and the speed slider of the robot are configured and sent to the arm controller.

The last part of the main code consists of defining the MQTT topics for communicating with the Master controller and creating the thread that is going to handle incoming MQTT messages, and another new thread (”thread_read”) to handle the interrupts generated by the incoming RTDE messages from the arm. The main thread is occupied by the ”report_cur_pos()”, which publishes via MQTT the current joint arm positions if they have been changed in the last iteration.

Other important functions defined in the code is ”change_sdo()”, as is the one in charge to change the SDO of the arm, which moves the gripper through RTDE. Likewise, ”write_RTDE()” is in charge of sending the desired joint values to the arm through RTDE. ”on_connect()” is a MQTT helper function that subscribes to the desired topics, and ”on_message()” is the function that is called whenever there is a new MQTT message to process them and call the necessary functions to move the arm or the gripper if necessary.

4.3.3.- Mobile ER robot (MiR+UR)

The software architecture of the mobile ER robot is a bit more complex that the previously explained case in Section 4.3.2 for the static UR robot, as there are more nodes inside the closed network used to communicate the robotic parts. This network is managed by the MiR robot, which has a router in the 192.168.12.1 IP with a DHCP server. It is configured to give the 192.168.12.20 IP to the MiR robot computer, the 192.168.12.40 IP to the UR robotic arm computer, the 192.168.12.30 IP to the ER computer, and the 192.168.12.35 IP to the 5G gateway. All of them are connected through an internal switch by cable.

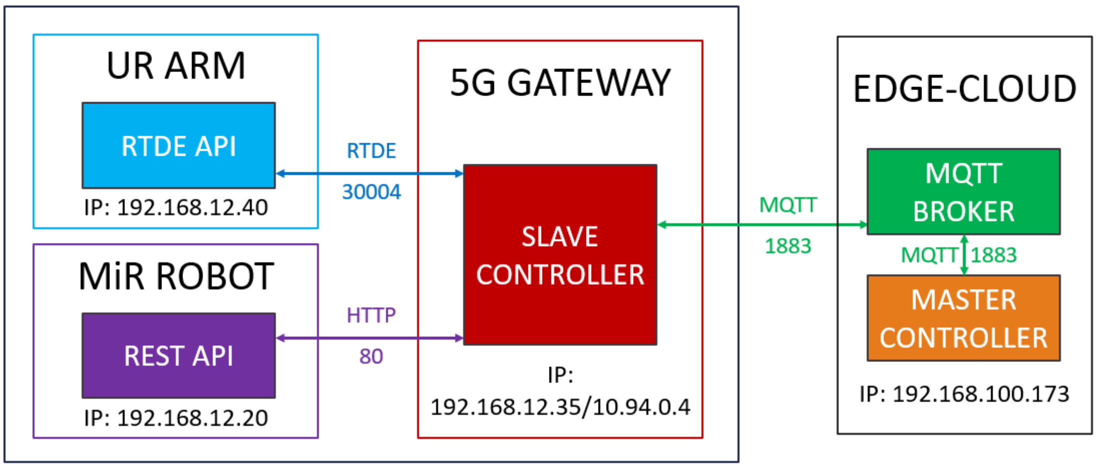


Figure 4.7.- Mobile ER robot software architecture

This network disposition permits the slave controller to communicate with the RTDE API running in the UR arm, and with the open REST API inside the MiR robot with the typical 80 TCP port for HTTP, as illustrated in Figure 4.7. The intended use case scenario for the MiR robot is to control it from the graphical interface of a web browser in the administrator's computer, but by using the internal REST API the robot can be controlled from code, and accessing it from the slave controller that is directly connected by cable to the API server, latency is significantly reduced.

Now that the software architecture has been reviewed, it is time to explain the developed code for the slave controller running in the 5G gateway of the mobile ER robot. The Python program is called ”mqtt_Slave_mobile.py”, and it is included in Annex A, at Section A.2.

The imports are the same as in the static case, but with three more additions. As the MiR REST API works with HTTP messages, it is needed to use the ”requests” and the ”urllib3” standartd libraries to handle these communications, and then there is a non-standard library called ”mir_api_s”, which has multiple functions to handle the REST API inside the MiR. It contains the necessary headers, request links and operations to communicate with the API, making the tasks of sending the mobile robot to a position much easier.

Next, the same variables are defined to control the robotic arm, with some additional ones to keep track of various states, to log data from the MiR robot, and more MQTT topics. The robotic arm configuration procedure is the same, but in this case commenting out the functions related with the SDO and the speed variables, as the robotic arm present in the mobile robot (UR5) did not support it.

The MQTT client is configured the same way, and has the same ”on_connect()” and ”on_message()” helper functions, and the functions for controlling the robotic arm are identical. However, there is another thread running the ”control_MiR()” function, which controls the general state and actions related with the mobile part of the robot. This function has a loop in which it checks for three things:

1. Whether there has been a new action posted for the MiR robot, in which case it sends a POST request to the API with the desired behaviour (moving to a checkpoint or making a relative move, for example).
2. It asks for the state of the MiR with the ”state_MiR()” function, to control if the mobile robot has finished with the assigned action (the MiR platform is in ”Ready” state after having posted the behaviour).
3. If the robot has not yet asked to drop the ball, is moving and the x-axis position is at the desired gripper release position, it will send a message to the Master controller to open the gripper. This position is calculated by using the exact position of the release (4.15 m in the x-axis), and the tray width (15 cm), to check if the robot is in the position with a +/- 7cm margin.

4.3.4.- Master Controller

The Master Controller is the code that acts as the Virtual PLC across the 5G network, and that runs in the Edge-Cloud Virtual Server represented in Figure 4.2. The developed Python program is called ”mqtt_Master_controller.py”, and it is included in Annex A, at Section A.3.

First of all, the libraries are imported. There are some generic ones, like ”math”, ”time”, ”logging”, ”threading” and ”socket”, which are very typical. In addition, the paho-mqtt library is also used to handle the MQTT messaging with the broker, and then the ”consts”, ”my_tools” and ”mir_api_s” previously developed libraries are also used.

Afterwards, there is a class that helps keep track of the different timestamps related with relevant timings. Many variables are also defined, some for the timestamp logging, others for the demo states, others for the MQTT topics, and some even for the robotic arm control.

Then, the usual functions for handling MQTT connections and messages are defined. Specially important is the ”on_message()” function, as some received messages can trigger changes of states (for example, that the gripper has been released or that the robot has reached a certain position) that affect the operation of the main program. Several functions are defined as well to trigger the gripper (”move_gripper()”), send a new position to the MiR robot (”move_mir()”), move both arms (”move_arms()”) or move them individually (”move_arm1()” and ”move_arm2()”). In addition, there are two relevant functions, as ”wait_pos()” is the function that waits on both robotic arms to reach their position in order to not continue with the execution of the use case, and ”write_data()” logs the gripper control timings to a CSV file.

Finally, ”run()” encapsulates the main loop of the controller’s program, and therefore the industrial use case actions. The step enumeration matches the ”demo_counter” variable.

0. The gripper is closed in order to grip the object, and move the arms into a safe position.
1. Once the arms are in position, the mobile robot is sent to the starting position for the relative move.
2. When the robot has reached the coordinates, the arms are moved into the needed position for the exchange.

3. As soon as the arms have finished moving, the mobile robot starts moving 2 metres in a straight line at a speed of 0.5 m/s.
4. Once the mobile robot has entered the defined exchange area, a message is sent to the gripper to release the object.
5. When the mobile robot has finished moving, a message is sent to the gripper to be in a relaxed state, and both arms are retracted into a safe position. Then, the recorded gripper timestamps are stored in a CSV file with the ”write_data()” function.
6. Finally, as soon as the the arms are in position, the mobile MiR robot is sent to a finishing position, and the program is closed once it arrives.

4.3.5.- MQTT broker implementation

The configuration file for the Mosquitto MQTT broker, found at ”/etc/mosquitto/mosquitto.conf”, is shown next. It is very basic, as it only has a few lines. The ”pid_file” ensures that there is a record of the broker’s process ID, which helps in management. Then, the ”persistance” variable is set to true, and the place to save the data is specified. This ensures that client sessions and messages are retained even if the broker restarts. Finally, the destination log file of the broker and the directory for additional configurations are stated.

```
pid_file /var/run/mosquitto.pid
persistence true
persistence_location /var/lib/mosquitto/
log_dest file /var/log/mosquitto/mosquitto.log
include_dir /etc/mosquitto/conf.d
```

If the additional configurations folder is searched, a file would appear at ”/etc/mosquitto/conf.d/default.conf”. Inside this file there are just two directives as shown next, stating that every user has to log into the broker (no anonymous clients are allowed), and that the file containing the encrypted user and password combinations is located in ”/etc/mosquitto/passwd”.

```
allow_anonymous false
password_file /etc/mosquitto/passwd
```

4.3.6.- Message sequence diagrams

This subsection intention is to illustrate the functional control logic implemented by showcasing the message exchange between the different network elements represented in Figure 4.2.

Runtime execution starts when the Cloud controller sends an MQTT message to the Slave running in the 5G gateway of the mobile robot, as showcased in Figure 4.8. This message triggers an HTTP POST message to the MiR internal server, so that an specific mission related with movement is started. In the positioning case, it is just a matter of sending the robot to a specific checkpoint on the map, but for the gripper event case (the one represented), a relative move of 2 metres in a straight line at 0.5 m/s is requested.

Afterwards, the Slave ensures to constantly poll the MiR REST API in order to get both the state and the position of the robot. That way, the program can determine whether the robot has already moved to the desired position and it is ready for the next step. In this case, once the robot passes the indicated zone for the object dropping, the slave sends an MQTT message to the controller, that then sends another message to the static robotic arm to release the gripper through the set Standard Digital Output RTDE message. Finally, when the robot reaches the final destination and switches from the ”Executing” to the ”Ready” state, another MQTT message is sent to the virtual controller to continue with the next step of the demonstration.

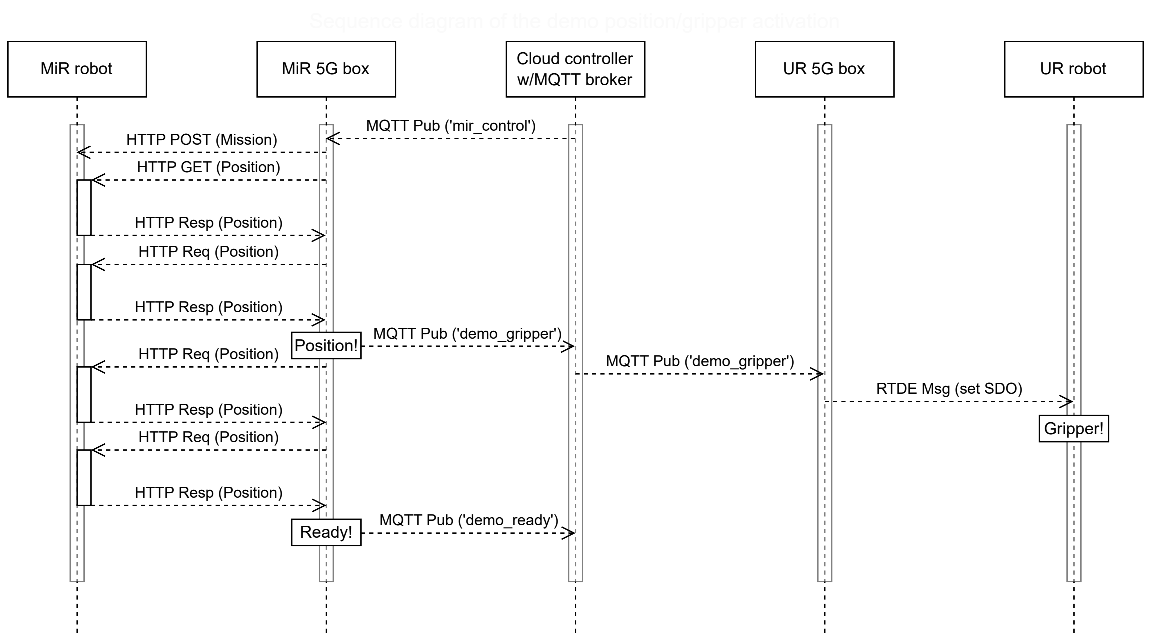


Figure 4.8.- MiR-Cloud-UR message sequence diagram

The other diagram in Figure 4.9 shows the sequence diagram of the messages exchanged by the controller with the UR arms in order to control their position. Both arms have the same control schema, so it is enough to explain one of them. Everything starts sending the requested values for each of the joints of both arms. This message generates on the respective 5G box a RTDE message to change the robot's registers for the target position. Both arms constantly send the current position of each joint at a specific rate previously defined through RTDE, independently of the rest of the messages. If the desired and actual position do not match (within a threshold), the Slave will keep sending to the Master controller a message with the current joint values, until they match inside the given threshold, in which case the Slave sends a message indicating that it is ready for the next step. Once the Cloud controller receives both ready messages, it continues with the demonstration loop.

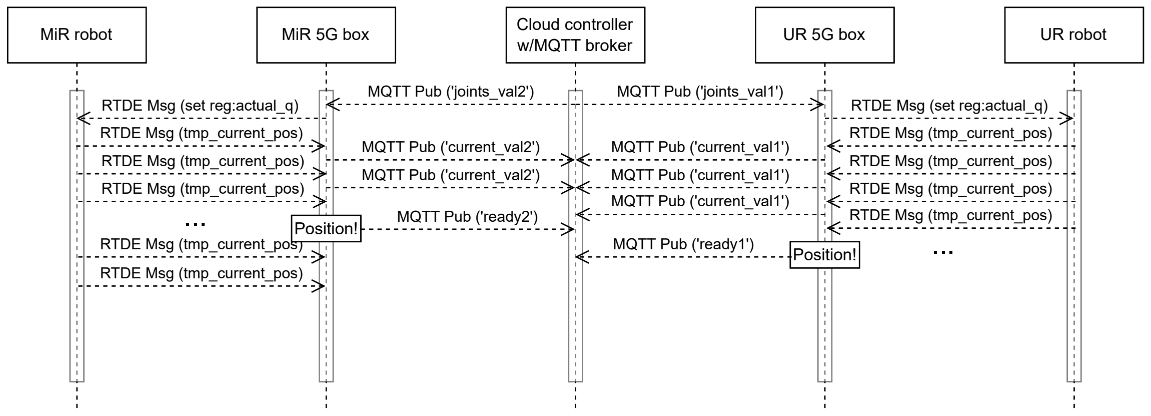


Figure 4.9.- UR-Cloud-UR message sequence diagram

4.4.- Operational KPIs for the demonstration execution

Once the idea for the industrial use case was established and developed, the related KPIs considered for its evaluation can be defined:

- KPI1 ”Operational Closed Control Loop Latency (OCCLL)”: this KPI would refer to the time it takes the announcement message from the 5G gateway in the mobile robot takes to arrive to the 5G gateway on the static robot. In the description of the Master Controller in Subsection 4.3.4, this would refer to step 4. The maximum allowed time, as described in Section 4.1, is 30 ms to ensure a correct operation of the use case. To better analyze the whole process, it can be divided into three timed stages, which are: the transmission time of the MQTT message from the 5G gateway of the mobile robot to the Cloud Controller (Ttx), the processing time at the controller (Tpr), and the time it takes the MQTT message to go from the controller to the 5G gateway of the static robot (Trx). The representation of these messages is depicted in Figure 4.8 as both MQTT Pub (’demo_gripper’) messages. This is how the resulting equation would be:

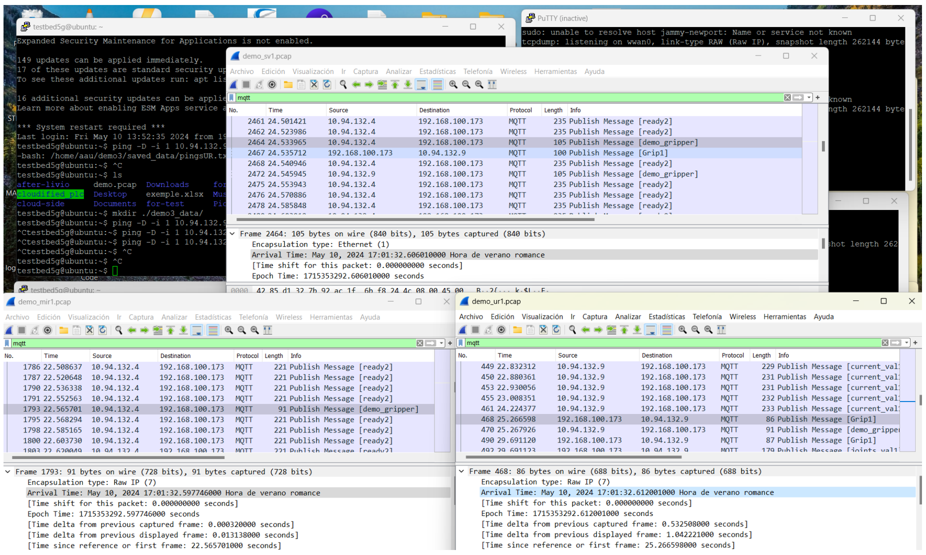
$$OCCLL[ms] = Ttx[ms] + Tpr[ms] + Trx[ms]$$

- KPI2 ”Percentage of Successful Iterations (PSI)”: out of all the iterations executed, determine the percentage of successful ones. The equation is just divide the number of successful runs (Nsuc) by the number of total iterations (Ntot), and multiplying by 100.

$$PSI[\%] = (Nsuc/Ntot) * 100$$

To be able to extract meaningful results out of the use case, it would require to run it a relevant number of times to generate meaningful statistics. Industrial use cases demand very high reliability (99.99% at least), but testing time for the thesis project was very limited due to operational constrains within the research lab, so 20 different executions would be the minimum allowable amount.

However, in order to ensure that those tests were accurate and to extract more data, network traffic was captured in all three devices involved in the use case (the two 5G gateways and the cloud controller). This, combined with ping values while the measurements were running, allowed to have more collected data, that later translated into benefits like assessing synchronization errors, checking if there were lost packets, and verifying the logged results against those recorded in the captures, like the ones illustrated in Figure 4.10.


testbed5g@ubuntu: ~
PuTTY (inactive)
demo_sv1.pcap
demo_mir1.pcap
demo_ur1.pcap
Archivo Edición Visualización Ir Captura Analizar Estadísticas Telefonía Wireless Herramientas Ayuda
mqtt

```
Expanded Security Maintenance for Applications is not enabled.

149 updates can be applied immediately.
17 of these updates are standard security up
To see these additional updates run: apt lis

16 additional security updates can be applie
Learn more about enabling ESM Apps service a

*** System restart required ***
Last login: Fri May 10 13:52:35 2024 from 19
testbed5g@ubuntu:~$ ping -D -i 1 10.94.132.9
-bash: /home/aau/demo3/saved_data/pingsUR.tx
testbed5g@ubuntu:~$ ^C
testbed5g@ubuntu:~$ ls
after-livio      demo.pcap  Downloads     for
cloudified_plc   Desktop    exemple.xlsx  Mus
cloud-side       Documents  for-test      Pic
testbed5g@ubuntu:~$ mkdir ./demo3_data/
testbed5g@ubuntu:~$ ping -D -i 1 10.94.132.9
^Ctestbed5g@ubuntu:~$ ping -D -i 1 10.94.132
^Ctestbed5g@ubuntu:~$ ping -D -i 1 10.94.132
^Ctestbed5g@ubuntu:~$ ^C
testbed5g@ubuntu:~$ ^C
testbed5g@ubuntu:~$
```

```
sudo: unable to resolve host jammy-newport: Name or service not known
tcpdump: listening on wwan0, link-type RAW (Raw IP), snapshot length 262144 byte
```

```
No.  Time       Source           Destination      Protocol Length Info
2461 24.501421  10.94.132.4      192.168.100.173  MQTT     235 Publish Message [ready2]
2462 24.523986  10.94.132.4      192.168.100.173  MQTT     235 Publish Message [ready2]
2464 24.533965  10.94.132.4      192.168.100.173  MQTT     105 Publish Message [demo_gripper]
2467 24.535712  192.168.100.173  10.94.132.9      MQTT     100 Publish Message [Grip1]
2468 24.540946  10.94.132.4      192.168.100.173  MQTT     235 Publish Message [ready2]
2472 24.545945  10.94.132.9      192.168.100.173  MQTT     105 Publish Message [demo_gripper]
2475 24.553943  10.94.132.4      192.168.100.173  MQTT     235 Publish Message [ready2]
2476 24.570886  10.94.132.4      192.168.100.173  MQTT     235 Publish Message [ready2]
2478 24.585848  10.94.132.4      192.168.100.173  MQTT     235 Publish Message [ready2]

Frame 2464: 105 bytes on wire (840 bits), 105 bytes captured (840 bits)
  Encapsulation type: Ethernet (1)
  Arrival Time: May 10, 2024 17:01:32.606010000 Hora de verano romance
  [Time shift for this packet: 0.000000000 seconds]
  Epoch Time: 1715353292.606010000 seconds
0000  42 85 d1 32 7b 92 ac 1f  6b f8 24 4c 08 00 45 00   B..2{... k.$L..E.
```

```
No.  Time       Source       Destination      Protocol Length Info
1786 22.508637  10.94.132.4  192.168.100.173  MQTT     221 Publish Message [ready2]
1787 22.520648  10.94.132.4  192.168.100.173  MQTT     221 Publish Message [ready2]
1790 22.536338  10.94.132.4  192.168.100.173  MQTT     221 Publish Message [ready2]
1791 22.552563  10.94.132.4  192.168.100.173  MQTT     221 Publish Message [ready2]
1793 22.565701  10.94.132.4  192.168.100.173  MQTT      91 Publish Message [demo_gripper]
1795 22.568294  10.94.132.4  192.168.100.173  MQTT     221 Publish Message [ready2]
1798 22.585165  10.94.132.4  192.168.100.173  MQTT     221 Publish Message [ready2]
1800 22.603730  10.94.132.4  192.168.100.173  MQTT     221 Publish Message [ready2]
1803 22.620049  10.94.132.4  192.168.100.173  MQTT     221 Publish Message [ready2]

Frame 1793: 91 bytes on wire (728 bits), 91 bytes captured (728 bits)
  Encapsulation type: Raw IP (7)
  Arrival Time: May 10, 2024 17:01:32.597746000 Hora de verano romance
  [Time shift for this packet: 0.000000000 seconds]
  Epoch Time: 1715353292.597746000 seconds
  [Time delta from previous captured frame: 0.000320000 seconds]
  [Time delta from previous displayed frame: 0.013138000 seconds]
  [Time since reference or first frame: 22.565701000 seconds]
```

```
No. Time       Source           Destination      Protocol Length Info
449 22.832312  10.94.132.9      192.168.100.173  MQTT     229 Publish Message [current_val
450 22.880361  10.94.132.9      192.168.100.173  MQTT     231 Publish Message [current_val
453 22.930056  10.94.132.9      192.168.100.173  MQTT     231 Publish Message [current_val
455 23.008351  10.94.132.9      192.168.100.173  MQTT     232 Publish Message [current_val
461 24.224377  10.94.132.9      192.168.100.173  MQTT     233 Publish Message [current_val
468 25.266598  192.168.100.173  10.94.132.9      MQTT      86 Publish Message [Grip1]
470 25.267926  10.94.132.9      192.168.100.173  MQTT      91 Publish Message [demo_gripper
490 29.691120  192.168.100.173  10.94.132.9      MQTT      87 Publish Message [Grip1]
492 29.691123  192.168.100.173  10.94.132.9      MQTT     179 Publish Message [joints_val

Frame 468: 86 bytes on wire (688 bits), 86 bytes captured (688 bits)
  Encapsulation type: Raw IP (7)
  Arrival Time: May 10, 2024 17:01:32.612001000 Hora de verano romance
  [Time shift for this packet: 0.000000000 seconds]
  Epoch Time: 1715353292.612001000 seconds
  [Time delta from previous captured frame: 0.532508000 seconds]
  [Time delta from previous displayed frame: 1.042221000 seconds]
  [Time since reference or first frame: 25.266598000 seconds]
```

Figure 4.10.- Evaluation of recorded traffic captures with Wireshark

4.5.- Industrial Testbed Development Conclusions

An outline of the outcomes of this chapter is done. The achieved results are aligned with the second objective set at the start of the thesis in Section 1.4, named ”OBJ2”: design and development of a industrial testbed including 5G wireless-synchronized behavior of robotic entities. The chapter starts with a description of how the idea for the industrial use case came along. Afterwards, the hardware architecture with the available devices at the AAU 5G Smart Lab is explained. Then, all the developed software and configuration files to achieve the use case are explained, along with architecture and message diagrams. Finally, the related KPIs for the performance evaluation of this use case are set.

5. Validation and Performance Evaluation Results

This chapter summarizes the performance results and observations to validate the implementation of the industrial use case.

5.1.- Validation

In order to validate the developed use case, some functional tests in industrial operational conditions were carried out at the AAU 5G Smart Production Lab.

In order to successfully run the industrial use case, some operational considerations have to be taken into account beforehand.

- The private 5G network of the university has to be online and all devices connected to it.
- Verify that the MiR robot is turned on and that it is in ”Ready” mode with green lights, as illustrated in Figure 5.1, not in ”Emergency Stop” with red lights or in ”Pause” mode with static yellow lights. In addition, it is a good practice to check in the GUI of the robot’s webpage whether the map is not calibrated with the robot’s position.

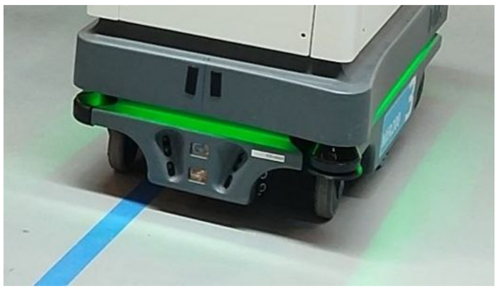
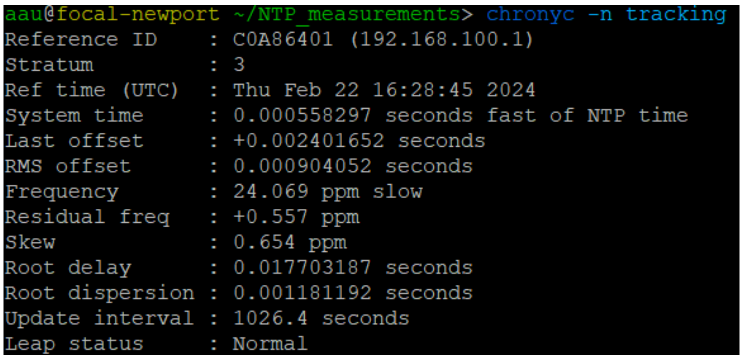


Figure 5.1.- MiR robot in Ready state (green lights)

- Make sure that the robotic arms are turned on and executing the ”slave_robot.urp” file, that they are in play and the speed is set to 30%.

- It is strongly recommended to use Python 3.9.X with virtual environments to install and use the required libraries and avoid incompatibilities. Remember to activate the virtual environments before running the code.
- Each of the computers running the programs have to be synchronized in time with the local NTP server. This can be checked through a simple query with Chrony as depicted in Figure 5.2. It has to be ensured that in the response the ”System time” value is not bigger than 1ms, as that might indicate that the clock is still adjusting, and that the timestamp value (”Ref Time (UTC)”) is not older than 2 minutes, as that might indicate that the clock lost the synchronization reference.



```
aau@focal-newport ~/NTP_measurements> chronyc -n tracking
Reference ID    : C0A86401 (192.168.100.1)
Stratum         : 3
Ref time (UTC)  : Thu Feb 22 16:28:45 2024
System time     : 0.000558297 seconds fast of NTP time
Last offset     : +0.002401652 seconds
RMS offset      : 0.000904052 seconds
Frequency       : 24.069 ppm slow
Residual freq   : +0.557 ppm
Skew            : 0.654 ppm
Root delay      : 0.017703187 seconds
Root dispersion : 0.001181192 seconds
Update interval : 1026.4 seconds
Leap status     : Normal
```

Figure 5.2.- Chrony NTP synchronization query

After ensuring all the previous considerations are met, it is time to run the code. First, it is needed to connect to each of the 5G gateways by using SSH and their assigned 5G IP (10.94.0.4 for the mobile robot, 10.94.0.9 for the static robot). Nowadays there is no need to use an external SSH client such as Putty, both Windows and Linux integrate it in their command terminals. The only information needed are the destination IP, the username (”aau” in this case), and the password, as it will be asked after sending the connection request. The resulting command would be ”ssh aau@10.94.0.4”. After successfully connecting to both 5G gateways of the static arm and the mobile robot, the Slave program has to be run (”mqtt_Slave_static.py” in the static arm and ”mqtt_Slave_mobile.py” in the mobile robot). Lastly, the Controller program (”mqtt_Master_controler.py”) is run in the Edge-Cloud server, acting as the virtual PLC that controls the slaves and the demo starts.

The first action is to retract the arms into a safe position. This step would relate with action point 0 of the enumeration described at Subsection 4.3.4. The actual message exchange during this move is described in Figure 4.9 Afterwards, as illustrated in Figure 5.3, the mobile robot gets into place with the MQTT commands sent by the controller, corresponding to step 1 of the Master Controller detailed in Subsection 4.3.4.

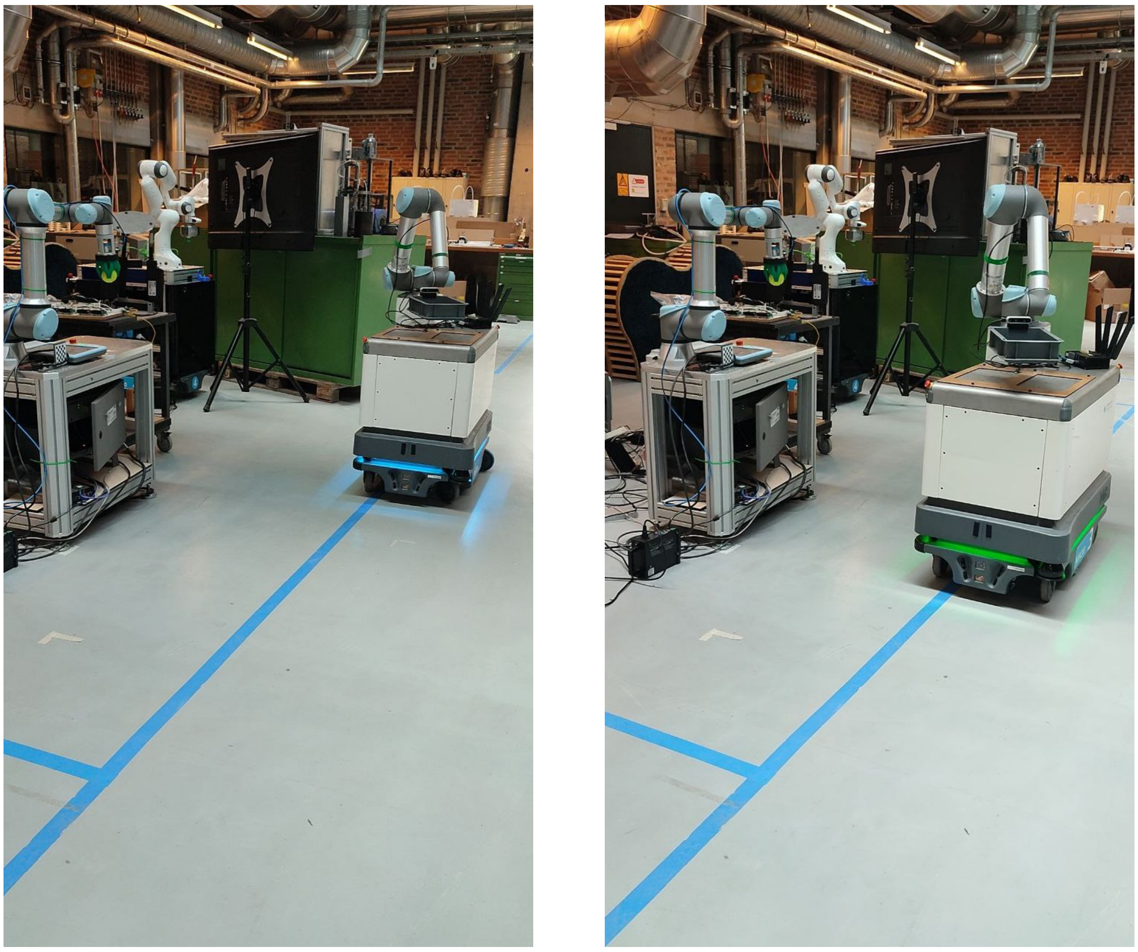
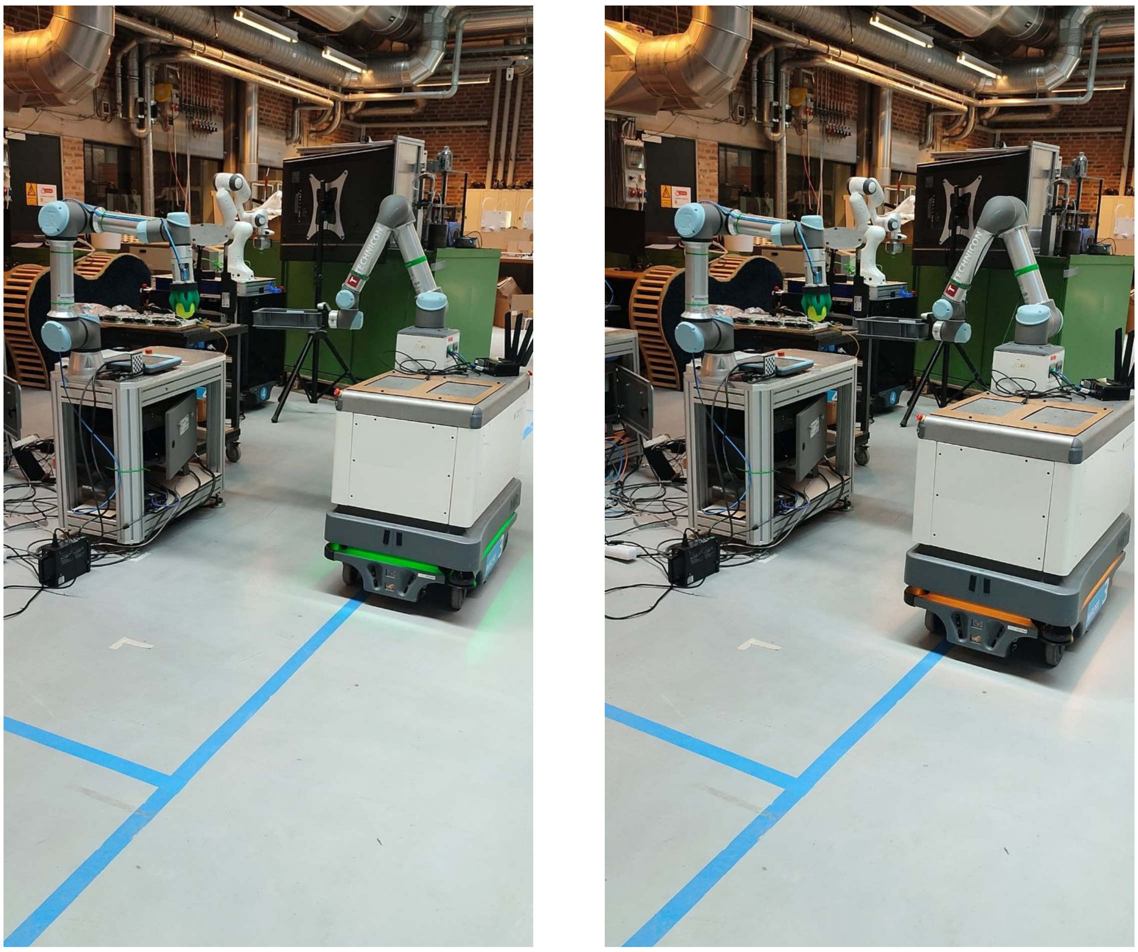

(a) Getting to the starting position (b) Arrived to the starting position

Figure 5.3.- Demo images preparation steps

The next step is to set the arms into the positions for the exchange as shown in Figure 5.4, related with step 2 defined in Subsection 4.3.4. Once the arms are in position, the described messages in Figure 4.8 start their flow by sending the POST request for a relative move to the MiR platform, and the robot starts moving in a straight line for 2 meters from the starting position, corresponding to step 3 of the Master Controller code.



(a) Moving the arms out (b) Starting the relative move

Figure 5.4.- Demo images initial steps previous to the gripper release

When the robot enters the exchange position area message, an MQTT message is sent to the cloud controller and then to the gripper to drop the ball, as described in the messages represented in Figure 4.8. This corresponds to step 4 of the Master Controller code explained in Subsection 4.3.4. If there was no problem during the execution of this step, the gripper should open in time and the ball would land on the tray, as depicted in Figure 5.5.

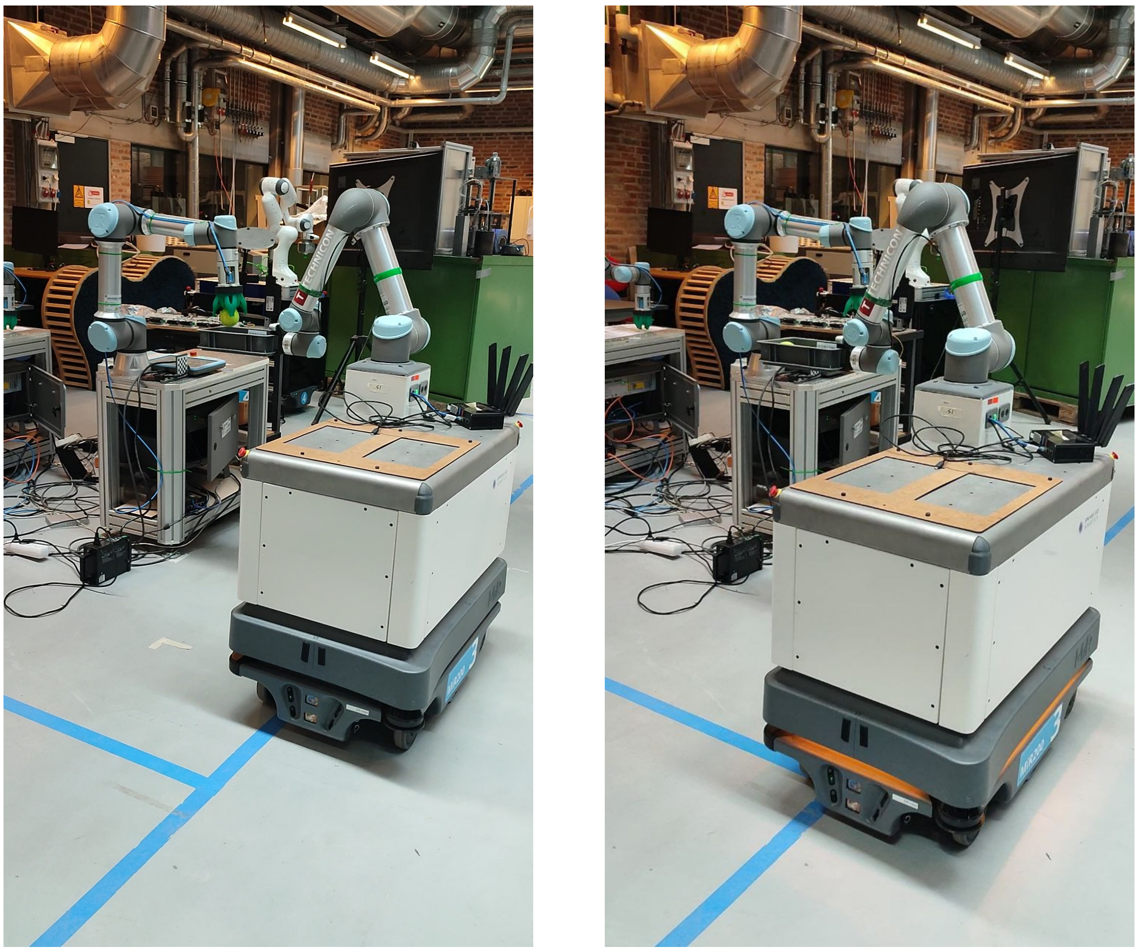
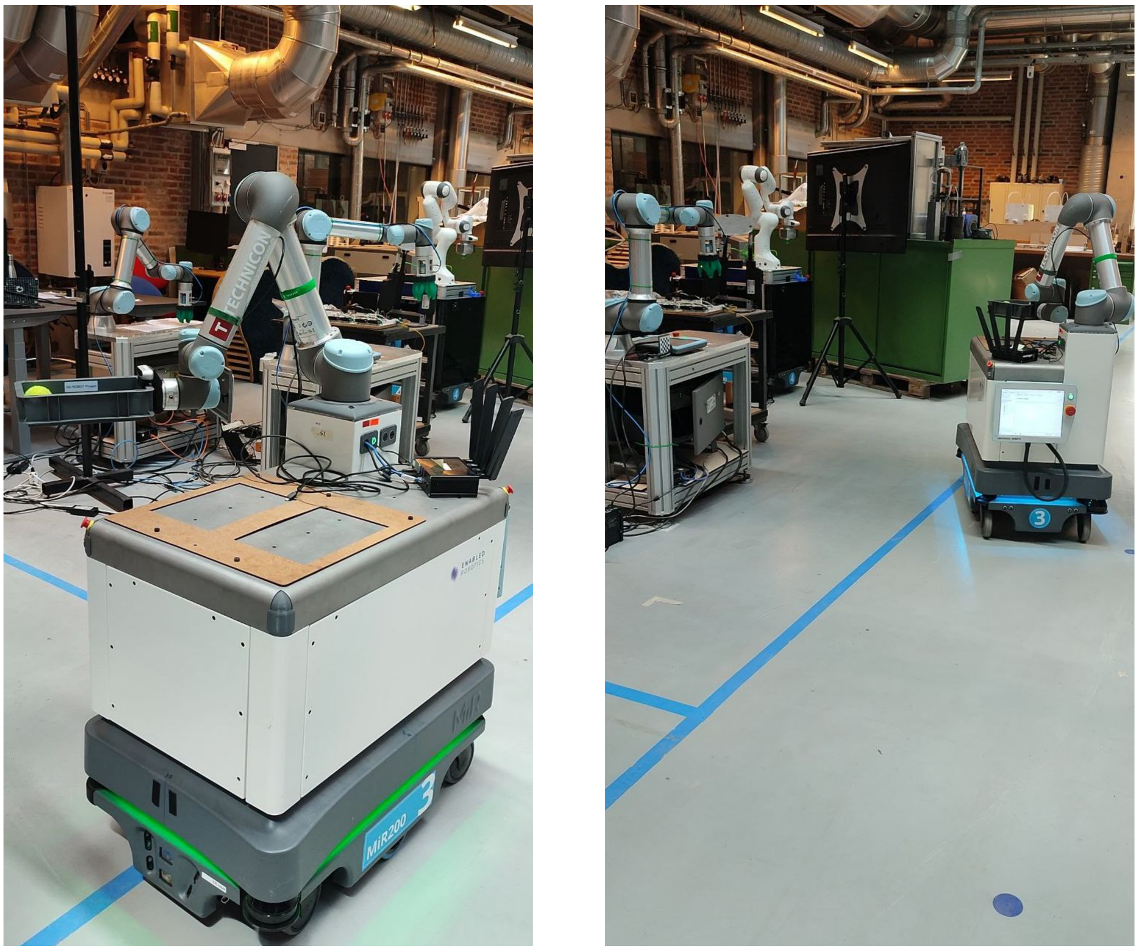

(a) Dropping the ball (b) The ball is on the tray

Figure 5.5.- Demo images from the gripper release event

Once the MiR platform finishes moving, the arms retract into a safe position, as illustrated in Figure 5.6, and finally, the mobile robot is sent to the finishing place by sending a new POST request to the internal MiR REST API. These steps are related with the fifth and last ones of the Master Controller execution that is showcased in Subsection 4.3.4.



(a) Retracting the arms into a safe position (b) Arriving to the final destination

Figure 5.6.- Demo images finishing steps

5.2.- Industrial use case performance evaluation results over 5G

Knowing that all clocks are synchronized as per the algorithms detailed in Chapter 2, there is no problem at comparing the timestamps from the different clocks to measure latency in 5G, as they are above the 1ms mark, and the achieved clock synchronization is mostly below a +-0.1ms.

5.2.1.- KPI1: Operational Closed Control Loop Latency

As discussed in Section 4.4, the KPIs defined for this use case aimed at evaluating the execution runtime performance for the gripper release event. First, the cycle time for the different use case components is analyzed.

Message transmission time (Ttx) is analyzed in Figure 5.7 in terms of Cumulative Distribution Function (CDF). As displayed, this runtime spans from 3.5 to 12 ms, with a median value of 7.7 ms. The measured transmission time performance from the 5G gateway on mobile robot to the Cloud server, representative of those signaling messages to trigger the ball passing action, is driven by the 5G uplink transmission performance. In 80% of the cases, Ttx is faster than 10 ms.

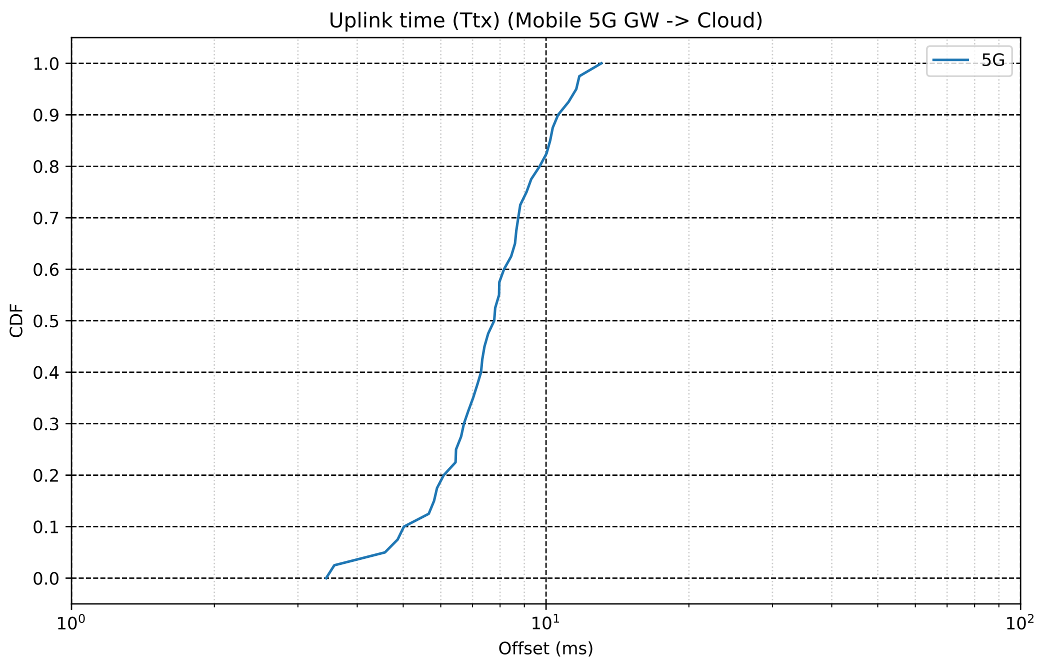


Figure 5.7.- Uplink latency CDF graph (representative of Ttx)

Similarly, the CDF for the transmission time from Cloud to the 5G gateway of the static robot (Trx), which is representative of the orders send from the Cloud server to the UR robotic arm on top of the mobile robot, is detailed in Figure 5.8. As illustrated, the performance is slightly more deterministic than in the Ttx case. Trx presents values spanning from 4 to 7 ms, with a median value of 5.7 ms. This different performance can be explained from the fact that Trx is dominated by 5G downlink transmission with is typically faster than the uplink one. This is due to the configured downlink/uplink scheduling ration configured in the 5G private network with is 7/3. Therefore, there are more air time slots allocated for Trx messages than for Ttx messages. Further, Ttx messages in uplink are not immediate, as the 5G network needs to issue an uplink permission for transmission, while downlink is generally instantaneous.

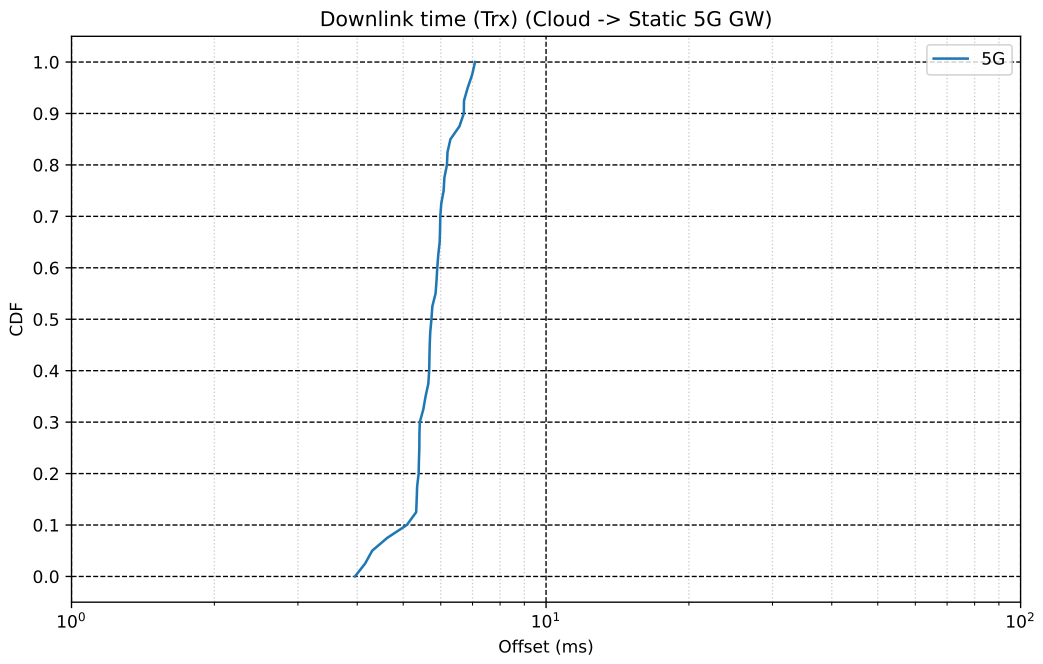


Figure 5.8.- Downlink latency CDF graph (representative of Trx)

The final contributor to the overall OCCLL is the processing time at the cloud (Tpr). The statistical distribution for Tpr is shown in Figure 5.9. As shown, processing time is really quick and lower than 2 ms in all cases.

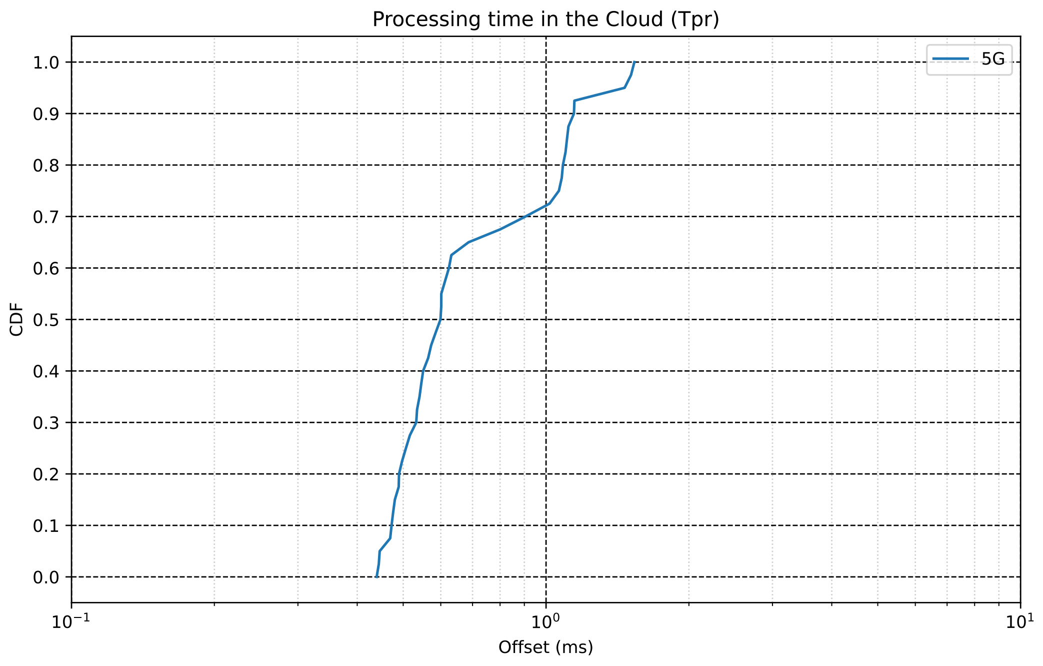


Figure 5.9.- Processing time in the cloud CDF graph (representative of Tpr)

The overall use case control loop latency (OCCLL) is analyzed in Figure 5.10. The combination of the different Ttx, Tpr, and Trx performance results in a distribution spanning from 8 to 19.9 ms, with a median value of 14 ms.

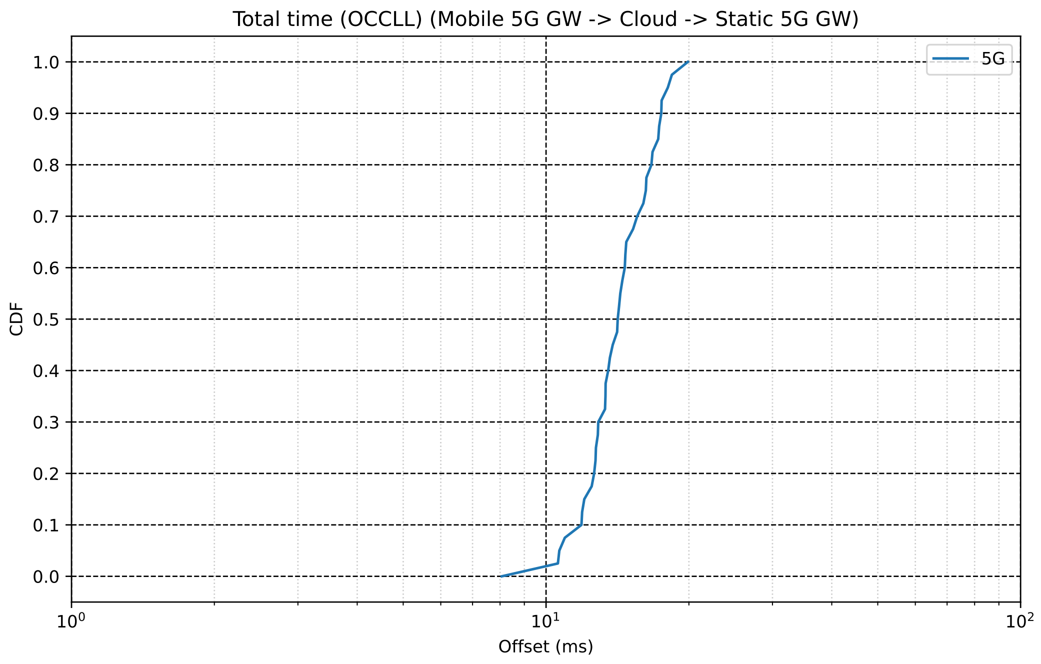


Figure 5.10.- Operational Closed Control Loop Latency CDF graph (OCCLL)

As previously discussed in the general requirements definition in Section 4.1, the maximum allowed time for the total OCCLL was defined in 30 ms. The measurements show that, even in the worst observed case of 19.9 ms, the industrial use case is viable over a 5G wireless network.

5.2.2.- KPI2: Percentage of Successful Iterations

While testing the demo over 5G, a run of 20 iterations was used to characterize the loop, the minimum amount stated at the KPI definition in Section 4.4. All of those executions had a favourable result at receiving the ball, translating into a 100% of successful actions. Even though this number of iterations are not enough to draw conclusions for an industrial case that needs to operate without any kind of stop during the entire year, the results are promising enough to encourage larger tests of reliability.

5.3.- Industrial use case performance evaluation results over WiFi 6E

While the 5G investigations and evaluations were performed, a question was raised: would the same synchronization and control scheme perform successfully over WiFi? Therefore, in order to verify this fact, similar implementations and tests were proposed using WiFi 6E in replacement of 5G.

In order to do that, some changes had to be made to the general architecture and devices used as seen in Figure 5.11. The 5G network was replaced by the Cisco SW9162 WiFi 6E Access Point, and the Gateworks gateways were changed by Intel NUCs 12 with their Intel AX210 WiFi 6E network card, because the gateways were having trouble with the driver of that network card. To give Internet access to the AP, as all the cabled network of the laboratory was tear down, a laptop was set up to share the ”eduroam” WiFi connection of the university using a direct cable to it. The entire network was configured to make use of the 6 GHz band offered in the latest version of the IEEE 802.11 protocol, in order to avoid interference and get the best performance.

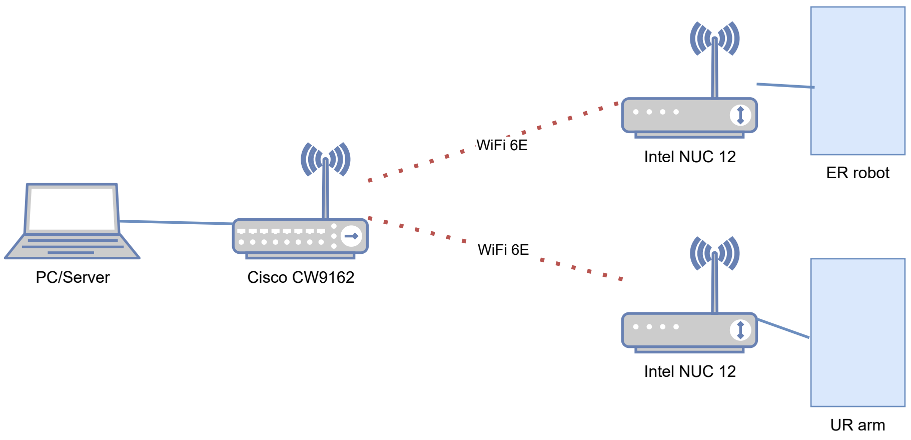


Figure 5.11.- WiFi network architecture diagram

As it was done with the 5G network, the OCCLL of the developed use case is measured. However, For WiFi, two different network scenarios were explored:

1. Best-case scenario (unloaded WiFi): only the two WiFi 6E modems for the industrial use case are connected in the network.
2. Worst-case scenario (loaded WiFi): apart from the robotic devices, an additional Intel NUC 12 to the WiFi network, and a big file is sent from the Server computer to it with SCP

while running the industrial application. This was done to simulate a user connected to the network that is demanding a lot of resources from the network, as it is the case, for example, when watching a high resolution video, or downloading an update for a system. This is a common scenario in WiFi networks that share connection with industrial and office applications. The file was created by creating a zip file with 10 copies of Big Buck Bunny in FullHD surround [58], making it approximately a 9 GB file.

The last consideration to take into account is time synchronization. As illustrated in Figure 5.12, where synchronization accuracy over WiFi 6E and over 5G are compared in terms of clock offset, a comparable synchronization level (within +/-0.1 ms) is experienced. It should be noted that the WiFi case presented a few odd spikes of up to 0.3-0.5 ms, which are caused by the aforementioned difference in air transmission access method.

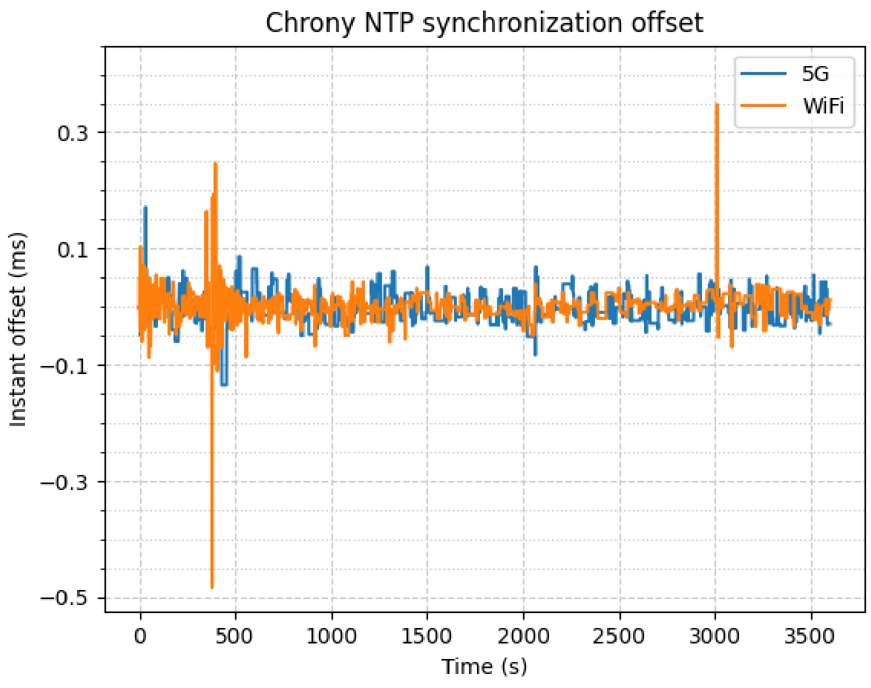


Figure 5.12.- Comparison of 5G and WiFi NTP synchronization

Knowing that all clocks are synchronized below the millisecond, the same method to measure latency as in 5G is used. In this case, 30 samples were taken for each of the two runs (one without the interference source and the other with it).

5.3.1.- KPI1: Operational Closed Control Loop Latency

The results depicted in Figure 5.13 show that Uplink transmission times (Ttx) over WiFi can be much faster than over 5G. Table 5.1 showcases the most relevant values. However, there is a really big difference between unloaded and loaded WiFi in the worst measured cases.

WiFi with interference does not show a big difference in the minimum values with respect to without it, but already the median value doubles the one unloaded, and in the 90%-ile value it is more than 10 times higher. Even worse, the maximum recorded value climbs up to the 174 ms mark, completely unacceptable for any URLLC scenario.

Comparing these results to 5G, unloaded WiFi performs much better. Across all measured values, WiFi without interference is 4 times faster than 5G. However, as discussed in the previous paragraph, interfered WiFi performed poorly. Even with the interference, WiFi can perform better that 5G in up to 70% of the cases, however, in the remaining 30%, the performance drastically worsens leading to Ttx values of up to 174 ms (e.g., 13 times worse than the maximum ones achieved over 5G).

Uplink time statistics (Ttx) (Mobile 5G GW -> Cloud)			
	5G	WiFi unloaded	WiFi loaded
Min [ms]	3.44	1.21	1.26
Mean [ms]	7.84	1.89	14.01
50%-ile [ms]	7.77	1.77	4.05
90%-ile [ms]	10.60	2.70	34.67
Max [ms]	13.07	3.17	174.16
Std [ms]	2.19	0.53	27.30

Table 5.1.- Uplink time statistics (Ttx) reference values

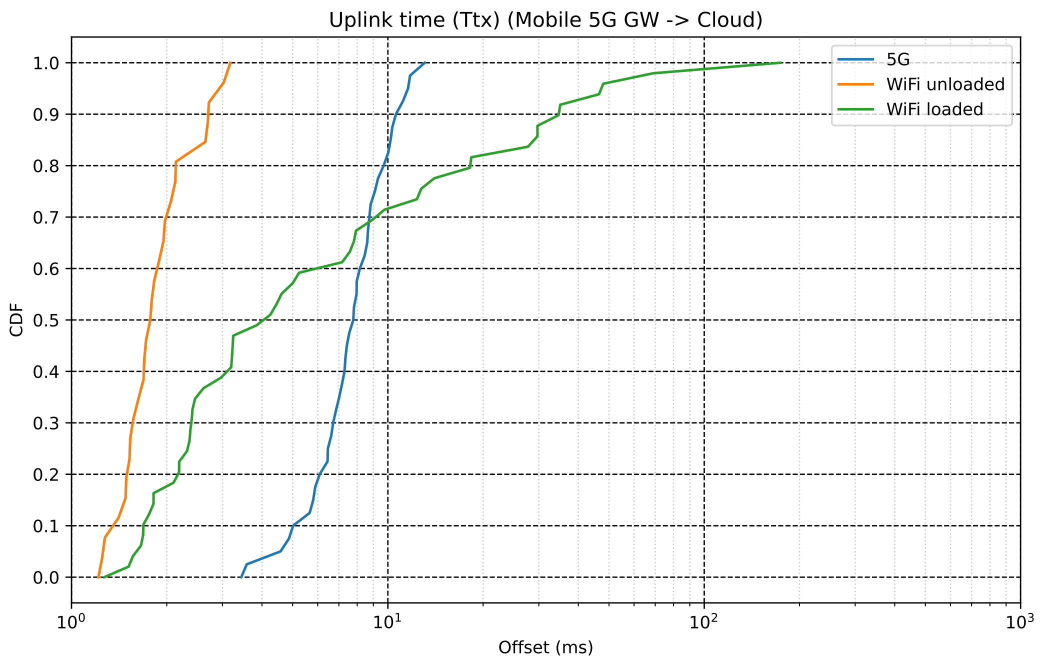


Figure 5.13.- Uplink latency CDF graph of all scenarios

Downlink reception times (Trx) times are illustrated in Figure 5.14, with relevant values referenced in Table 5.2, and they show much closer performance across all scenarios

WiFi unloaded performs again well, with a 50%-ile latency of 4.13 ms, although the maximum creeps up to 7.52 ms. Minimum WiFi values with interference, as in the uplink case, are not very different from the unloaded WiFi values, but the median value is again double than that without interference, in the 90%-ile it is more than 3 times higher, and the maximum is nearly 10 times worse.

Comparing these results to 5G, unloaded WiFi performs a better, but the difference is not as big as before. Across all measured values, WiFi measurements without interference are around 25% lower than those of 5G, except in the maximum one which is comparable. However, as discussed in the previous paragraph, worst-case WiFi performed very bad again. This time, nearly all measurements with the loaded WiFi network were worse than in 5G. In the 50%-ile the latency is already double than over 5G, in the 90%-ile triple, and the maximum creeps up to more than 50 ms.

Downlink time (Trx) (Cloud -> Static 5G GW)			
	5G	WiFi unloaded	WiFi loaded
Min [ms]	3.95	2.92	3.72
Mean [ms]	5.74	4.37	12.04
50%-ile [ms]	5.73	4.13	9.93
90%-ile [ms]	6.70	5.62	16.88
Max [ms]	7.08	7.52	62.78
Std [ms]	0.68	1.16	9.58

Table 5.2.- Downlink time (Trx) reference values

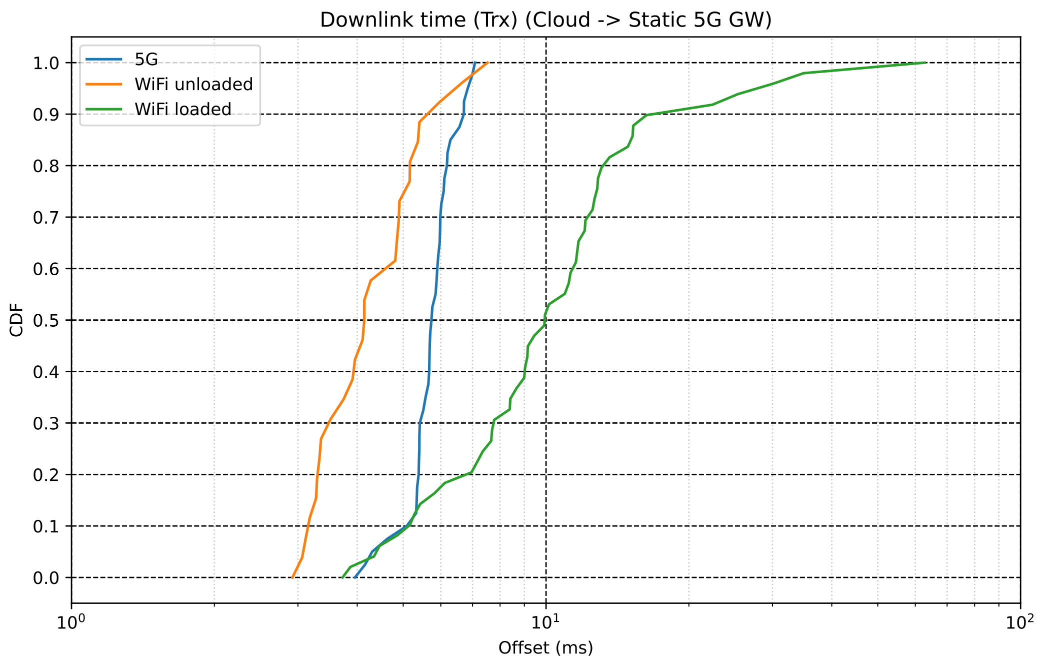


Figure 5.14.- Downlink CDF graph all scenarios

Processing time at the cloud server (Tpr), shown in Figure 5.15, is very similar across all measurements, as it is independent of the network used. Most of the results are below 1 ms, and the highest recorded maximum across all of them is just 1.53 ms. This indicates that none of the measured runs were affected by a big processing time, so the network is the most influential factor in the total latency time. Relevant reference values are summarized in Table 5.3.

Processing time in the Cloud (Tpr)			
	5G	WiFi unloaded	WiFi loaded
Min [ms]	0.43	0.26	0.08
Mean [ms]	0.74	0.55	0.60
50%-ile [ms]	0.59	0.48	0.62
90%-ile [ms]	1.14	0.89	1.02
Max [ms]	1.53	1.04	1.12
Std [ms]	0.32	0.23	0.31

Table 5.3.- Processing time in the Cloud (Tpr) reference values

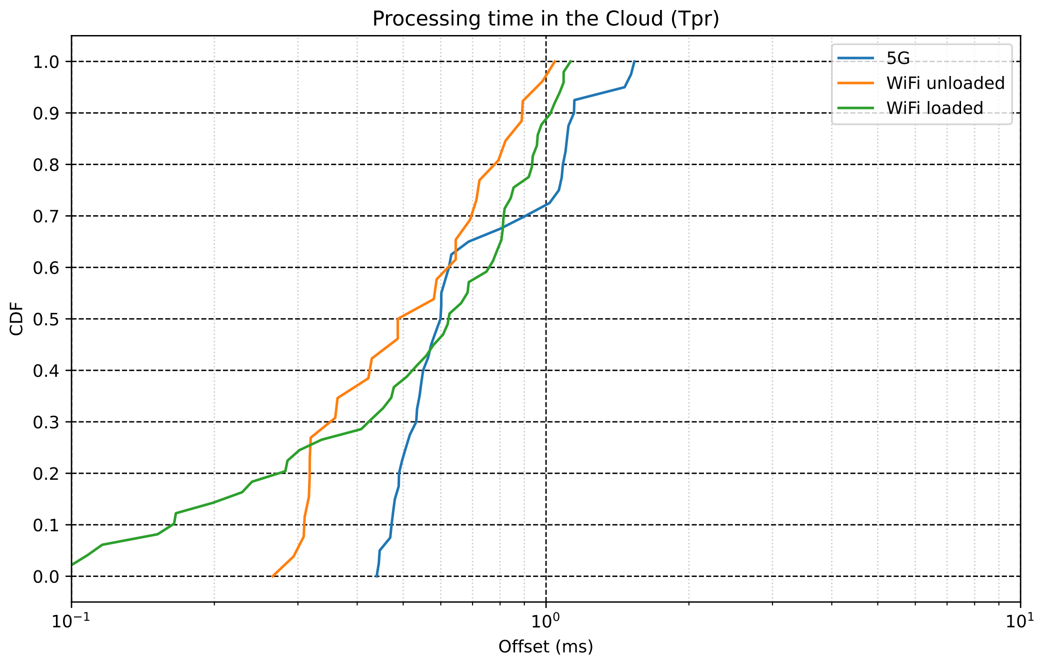


Figure 5.15.- Processing time in the cloud CDF graph all scenarios

Finally, Figure 5.13 represents the total latency of the use case, defined in the KPI as OCCLL. WiFi unloaded performs very well, as the 50%-ile of the total time is 6.71 ms, and even the maximum is less than 10 ms, as showcased in Table 5.4. However, latency in WiFi loaded is mostly more than double the one without interference. But even worse, upwards of the 75%-ile, the total time in loaded worst-case conditions climbs above the 30 ms mark, and the maximum reaches more 186 ms.

Comparing WiFi to 5G, unloaded WiFi performs considerably better than 5G, but in the loaded case performs very bad. Across all the measurements, the OCCLL of 5G is usually more than double of that of the case of unloaded WiFi. However, WiFi with interference is worse than 5G in most of the cases, and as exemplified in the downlink case, its performance gets drastically worse above the 75%-ile, specially in the worst-case scenario, as the maximum is nearly 10 times higher than the one with 5G.

Total time (OCCLL) (Mobile 5G GW -> Cloud -> Static 5G GW)			
	5G	WiFi unloaded	WiFi loaded
Min [ms]	8.06	4.70	6.92
Mean [ms]	14.33	6.82	26.66
50%-ile [ms]	14.15	6.71	16.70
90%-ile [ms]	17.49	8.28	52.76
Max [ms]	19.89	9.37	186.13
Std [ms]	2.44	1.22	28.48

Table 5.4.- Operational Closed Control Loop Latency (OCCLL) reference values

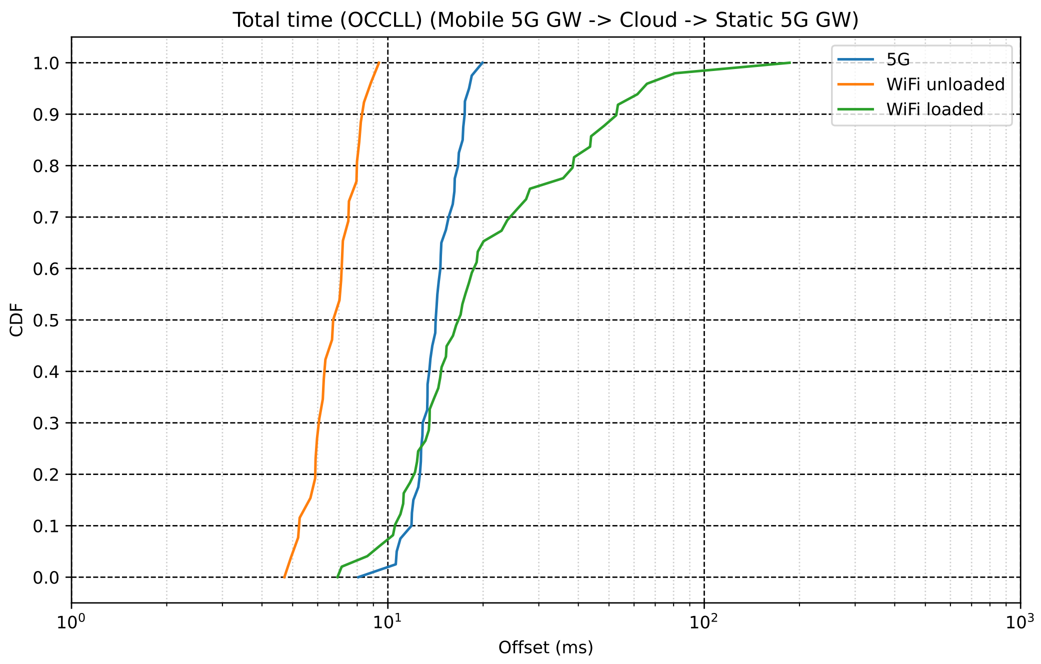


Figure 5.16.- Operational Closed Control Loop Latency CDF graph all scenarios

5.3.2.- KPI2: Percentage of Successful Iterations

Unfortunately, it was not possible to asses this KPI in the WiFi case due to operational constrains at the AAU 5G Smart Production Lab which was undergoing a renovation during the final testing phase, making the reference scenario slightly changed as compared to the original one explored with 5G.

5.4.- Validation and Performance Evaluation Conclusions

As an ending for this chapter, it is necessary to summarize the objectives and KPIs set. The fulfilled tasks are aligned with the third objective ”OBJ3”: validation and performance evaluation of the implemented 5G solution and comparison with alternative wireless access methods (e.g., WiFi 6E). This evaluation was carried out testing the developed industrial use case over both the private 5G network at the laboratory and a WiFi 6E network. In addition, the WiFi scenario was assessed both without and with interference from a

external device. In respect to the first KPI, the OCCLL, ideal WiFi without interference performed the best across all metrics (except processing time in the cloud, as it is not dependant on the network used). WiFi achieved approximately half of the latency of 5G in both uplink, downlink and total time. However, as already discussed in Section 1.2, WiFi is very susceptible from interference, and this was clearly showed in the completed tests with a crowded network. Performance dropped massively, in all cases worse than 5G. In the worst 10% of the situations, the total latency was more than 50 ms, and in the worst 1%, it was more than 100 ms. The results show that, even if WiFi 6E can offer the best performance, unless it is satisfied that the WiFi network will always run completely free of disturbances (either from its own or external networks), it is not as suitable for the URLLC industrial use cases as 5G.

6. Conclusions and future work

6.1.- Conclusions

This thesis dealt with integration of wireless networks into robotics within the fourth industrial revolution (Industry 4.0) context. By leveraging wireless, industrial manufacturing use cases and applications can enhance their flexibility, quality, efficiency, increase safety and reliability.

In particular, in this work, the main objective was the investigation and validation of wireless architectures and protocols for the synchronization and control of industrial robots. Being able to synchronize multiple robots in a wireless fashion will bring industry to the next level, as this will enable the coordination of mobile and static elements. In order to achieve this, the work was structured in 3 phases.

The objective OB1, related with the ”investigation and characterization of time synchronization in 5G networks considering static and mobile industrial use case conditions” was analyzed in Chapter 3. In it, the time synchronization of a 5G gateway on a mobile robot was analyzed, both in static and mobile cases. The first tests with the baseline configuration of the NTP client were not satisfactory for URLLC use cases, as the synchronization offset had regular spikes over the +/-1 ms mark. However, the results of the configuration optimization in the 5G gateway did achieve synchronization offsets below the +/-0.1 ms mark on 99.9% of the time, achieving the KPI objective set. In addition, it was demonstrated that there is no difference to the time synchronization between static and mobile industrial cases (up to 1 m/s).

The following objective OB2, related with the ”design and development of a industrial testbed including 5G wireless-synchronized behavior of robotic entities” was described in Chapter 4. The development process is detailed, from the initial idea to make a industrial use case scenario, going through the specification and programming of the different network and robotic elements used, and describing the messages and the KPI that are involved in the use case.

The last objective, OB3, related with the ”validation and performance evaluation of the implemented 5G solution and comparison with alternative wireless access methods (e.g., WiFi 6E)” was analyzed in Chapter 5. Both 5G and WiFi 6E over the 6 GHz band were tested. WiFi 6E managed to achieve approximately half of the latency of 5G in both uplink, downlink and total time, when operated in ideal conditions (e.g., full control of connected devices and no background traffic), going from a mean of 14.33 ms in 5G to just 6.82 ms in OCCLL (total time). However, it has to be taken into account the susceptibility of WiFi to interference, as the for the same statistic, it can worsen up to 52.76 ms in the 90% worst percentile, and a maximum of even 186 ms. That is why WiFi is not suitable for URLLC UCs unless it can be ensured that both the network and the radio-electric band are going to be reserved and monitored to perform at its best, as otherwise 5G offers the most reliable performance, with bounded OCCLL values of maximum 19.89 ms.

6.2.- Future work

A number of actions are proposed for potential continuation and optimization of the presented work.

6.2.1.- Better time synchronization

Even though the achieved synchronization levels were good enough for the requirements of the project, there is still room for improvement. There is the possibility that NTP results could still be enhanced with a bit more of tweaking, specially if the server gets the time reference signal from a GPS signal. But most likely, the biggest improvement would be to change the update protocol to PTP, even if that requires the use of more specific hardware.

6.2.2.- Centralised controller functions

The original implementation of having a slave controller that told the controller when it was in the desired position worked well, but another implementation was also thought. The diagram is shown in Figure 6.1, and it is more aligned with the idea of having a very simple slave program on the 5G boxes, and implementing the functionality in the cloud controller. Instead of the MiR 5G gateway being the computer in charge of taking the decision on when the robot is at the desired

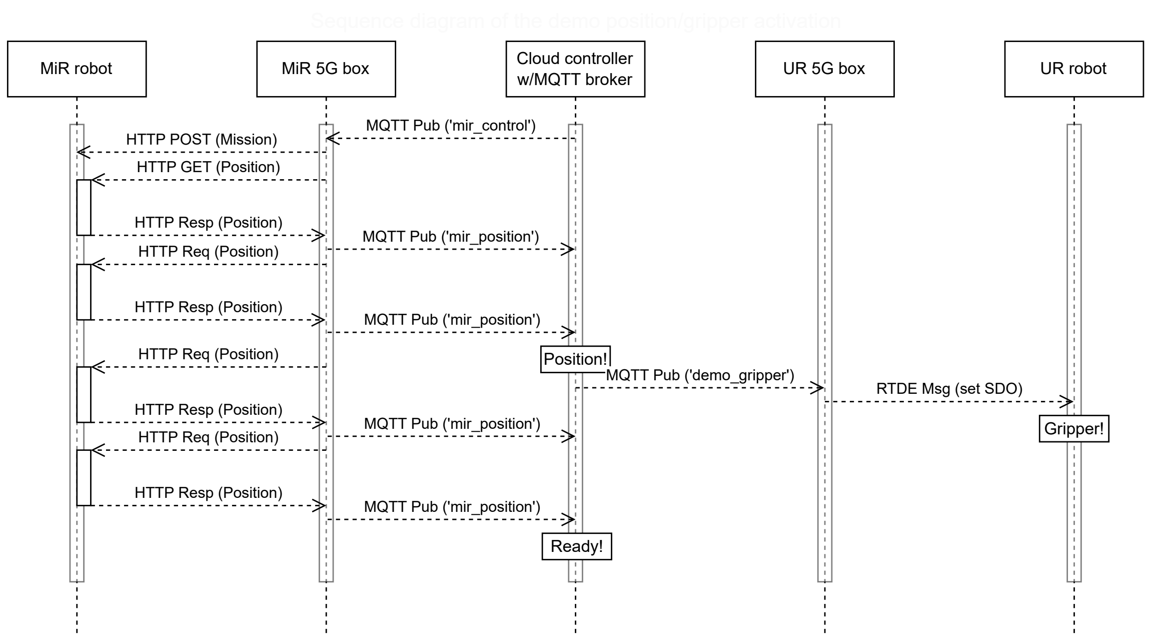


Figure 6.1.- Centralised controller MiR-Cloud-UR message sequence diagram

position, it would be the Cloud controller the one in charge. This behaviour puts more load to the network, but it should not be as big to have a significant impact to the performance of the demonstration, and it makes implementing new functionality or just the reconfiguration of the demo much easier.

6.2.3.- Intel RealSense camera streaming

Even though it was out of the scope of this thesis, it was very interesting to test the video and depth streaming over the 5G network. In my opinion, this is an interesting line of investigation, as Computer Vision becomes more prevalent in the robotic field and devices become lighter, so that the processing is offloaded to the Cloud and specially Edge-Cloud.

Bibliography

[1] *Top 15 New Technology Trends In [2024] – Dev Technosys*. en. Section: AI Development. July 2022. URL: https://devtechnosys.com/insights/new-technology/ (visited on 07/13/2024).

[2] *Industrial Revolutions through the Ages | Simio*. URL: https://digital-twin.simio.com/digital-twin-industry-40/industrial-revolution-through-the-ages.php (visited on 07/13/2024).

[3] Baotong Chen et al. “Smart Factory of Industry 4.0: Key Technologies, Application Case, and Challenges”. In: *IEEE Access* 6 (2018). Conference Name: IEEE Access, pp. 6505–6519. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2017.2783682. URL: https://ieeexplore.ieee.org/document/8207346 (visited on 07/13/2024).

[4] Martin Wollschlaeger, Thilo Sauter, and Juergen Jasperneite. “The Future of Industrial Communication: Automation Networks in the Era of the Internet of Things and Industry 4.0”. In: *IEEE Industrial Electronics Magazine* 11.1 (Mar. 2017). Conference Name: IEEE Industrial Electronics Magazine, pp. 17–27. ISSN: 1941-0115. DOI: 10.1109/MIE.2017.2649104. URL: https://ieeexplore.ieee.org/abstract/document/7883994 (visited on 07/13/2024).

[5] M. Felser. “Real-Time Ethernet - Industry Prospective”. In: *Proceedings of the IEEE* 93.6 (June 2005). Conference Name: Proceedings of the IEEE, pp. 1118–1129. ISSN: 1558-2256. DOI: 10.1109/JPROC.2005.849720. URL: https://ieeexplore.ieee.org/document/1435742 (visited on 03/04/2024).

[6] *Modernización completa de sistemas eléctricos*. es-ES. URL: https://autecnia.com/all/nuestros-servicios/modernizaciones-y-migraciones-de-sistemas-electricos-y-de-control-antiguos/ (visited on 07/13/2024).

[7] Tomas Lennvall, Stefan Svensson, and Fredrik Hekland. “A comparison of WirelessHART and ZigBee for industrial applications”. In: *2008 IEEE International Workshop on Factory Communication Systems*. May 2008, pp. 85–88. DOI: 10.

1109/WFCS.2008.4638746. URL: https://ieeexplore.ieee.org/document/4638746 (visited on 07/13/2024).

[8] Juan Carlos Cano et al. “Evolution of IoT: An Industry Perspective”. In: *IEEE Internet of Things Magazine* 1.2 (Dec. 2018). Conference Name: IEEE Internet of Things Magazine, pp. 12–17. ISSN: 2576-3199. DOI: 10.1109/IOTM.2019.1900002. URL: https://ieeexplore.ieee.org/document/8717596 (visited on 07/13/2024).

[9] Mattia Rizzi et al. “Using LoRa for industrial wireless networks”. In: *2017 IEEE 13th International Workshop on Factory Communication Systems (WFCS)*. May 2017, pp. 1–4. DOI: 10.1109/WFCS.2017.7991972. URL: https://ieeexplore.ieee.org/document/7991972 (visited on 07/13/2024).

[10] Bernd Holfeld et al. “Wireless Communication for Factory Automation: an opportunity for LTE and 5G systems”. In: *IEEE Communications Magazine* 54.6 (June 2016). Conference Name: IEEE Communications Magazine, pp. 36–43. ISSN: 1558-1896. DOI: 10.1109/MCOM.2016.7497764. URL: https://ieeexplore.ieee.org/document/7497764 (visited on 07/13/2024).

[11] Mina Rady et al. “How does Wi-Fi 6 fare? An industrial outdoor robotic scenario”. In: *Ad Hoc Networks* 156 (Apr. 2024), p. 103418. ISSN: 1570-8705. DOI: 10.1016/j.adhoc.2024.103418. URL: https://www.sciencedirect.com/science/article/pii/S1570870524000295 (visited on 07/13/2024).

[12] *Private 5G use cases for industry 4.0*. en-GB. URL: https://stlpartners.com/articles/private-cellular/private-5g-use-cases/ (visited on 07/13/2024).

[13] Ignacio Rodriguez et al. “5G Swarm Production: Advanced Industrial Manufacturing Concepts Enabled by Wireless Automation”. In: *IEEE Communications Magazine* 59.1 (Jan. 2021). Conference Name: IEEE Communications Magazine, pp. 48–54. ISSN: 1558-1896. DOI: 10.1109/MCOM.001.2000560. URL: https://ieeexplore.ieee.org/document/9356516/authors#authors (visited on 07/08/2024).

[14] Jorge Navarro-Ortiz et al. “A Survey on 5G Usage Scenarios and Traffic Models”. In: *IEEE Communications Surveys & Tutorials* 22.2 (2020). Conference Name: IEEE Communications Surveys & Tutorials, pp. 905–929. ISSN: 1553-877X. DOI: 10.1109/COMST.2020.2971781. URL: https://ieeexplore.ieee.org/document/8985528 (visited on 07/15/2024).

[15] *5G Applications and Use Cases*. en-US. URL: https://www.digi.com/blog/post/5g-applications-and-use-cases (visited on 07/15/2024).

[16] Vikash Mishra, Debabrata Das, and Namo Narayan Singh. “Novel Algorithm to Reduce Handover Failure Rate in 5G Networks”. In: *2020 IEEE 3rd 5G World Forum (5GWF)*. Sept. 2020, pp. 524–529. DOI: 10.1109/5GWF49715.2020.9221410. URL: https://ieeexplore.ieee.org/document/9221410 (visited on 07/15/2024).

[17] Preben Mogensen and Ignacio Rodriguez. “5G for Smart Production”. en. In: *The Future of Smart Production for SMEs: A Methodological and Practical Approach Towards Digitalization in SMEs*. Ed. by Ole Madsen et al. Cham: Springer International Publishing, 2023, pp. 327–333. ISBN: 978-3-031-15428-7. DOI: 10.1007/978-3-031-15428-7_28. URL: https://doi.org/10.1007/978-3-031-15428-7_28 (visited on 07/17/2024).

[18] Steve Shellhammer, Alfred Asterjadhi, and Yanjun Sun. “Overview of IEEE 802.11”. In: *IEEE 802.11ba: Ultra-Low Power Wake-up Radio Standard*. Conference Name: IEEE 802.11ba: Ultra-Low Power Wake-up Radio Standard. IEEE, 2023, pp. 9–24. ISBN: 978-1-119-67099-5. DOI: 10.1002/9781119671015.ch2. URL: https://ieeexplore.ieee.org/document/10017396 (visited on 07/15/2024).

[19] M. Bertocco, G. Gamba, and A. Sona. “Is CSMA/CA really efficient against interference in a wireless control system? An experimental answer”. In: *2008 IEEE International Conference on Emerging Technologies and Factory Automation*. ISSN: 1946-0759. Sept. 2008, pp. 885–892. DOI: 10.1109/ETFA.2008.4638501. URL: https://ieeexplore.ieee.org/document/4638501 (visited on 07/13/2024).

[20] Rony Kumer Saha. “Coexistence of Cellular and IEEE 802.11 Technologies in Unlicensed Spectrum Bands -A Survey”. In: *IEEE Open Journal of the*

Communications Society 2 (2021). Conference Name: IEEE Open Journal of the Communications Society, pp. 1996–2028. ISSN: 2644-125X. DOI: 10.1109/OJCOMS.2021.3106502. URL: https://ieeexplore.ieee.org/document/9520660 (visited on 07/15/2024).

[21] Pejman Roshan and Jonathan Leary. *802.11 Wireless LAN Fundamentals*. en. Google-Books-ID: 752JrPawu_sC. Cisco Press, 2004. ISBN: 978-1-58705-077-0.

[22] Andreas Fink et al. “Empirical Performance Evaluation of EnterpriseWi-Fi for IIoT Applications Requiring Mobility”. In: *European Wireless 2021; 26th European Wireless Conference*. Nov. 2021, pp. 1–8. URL: https://ieeexplore.ieee.org/abstract/document/9657101 (visited on 07/17/2024).

[23] *Open Systems Interconnection Model - an overview | ScienceDirect Topics*. URL: https://www.sciencedirect.com/topics/computer-science/open-systems-interconnection-model (visited on 07/16/2024).

[24] Chunju Shao et al. *IEEE 802.11 Medium Access Control (MAC) Profile for Control and Provisioning of Wireless Access Points (CAPWAP)*. Request for Comments RFC 7494. Num Pages: 13. Internet Engineering Task Force, Apr. 2015. DOI: 10.17487/RFC7494. URL: https://datatracker.ietf.org/doc/rfc7494 (visited on 07/16/2024).

[25] J. Postel. *Internet Protocol*. en. Tech. rep. RFC0791. RFC Editor, Sept. 1981, RFC0791. DOI: 10.17487/rfc0791. URL: https://www.rfc-editor.org/info/rfc0791 (visited on 07/15/2024).

[26] *Internet Control Message Protocol*. Request for Comments RFC 792. Num Pages: 21. Internet Engineering Task Force, Sept. 1981. DOI: 10.17487/RFC0792. URL: https://datatracker.ietf.org/doc/rfc792 (visited on 07/16/2024).

[27] Robert T. Braden. *Requirements for Internet Hosts - Communication Layers*. Request for Comments RFC 1122. Num Pages: 116. Internet Engineering Task Force, Oct. 1989. DOI: 10.17487/RFC1122. URL: https://datatracker.ietf.org/doc/rfc1122 (visited on 07/16/2024).

[28] *MQTT Version 3.1.1*. URL: https://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html (visited on 07/16/2024).

[29] David Arias-Cachero Rincón. "Kill-the-PLC: implementation and performance evaluation of a disruptive robotic cell environment based on 5G and edge-cloud technologies". eng. Accepted: 2023-07-28T07:58:38Z. MA thesis. July 2023. URL: https://digibuo.uniovi.es/dspace/handle/10651/69235 (visited on 07/16/2024).

[30] *What is Industry 4.0 and how does it work? | IBM*. en-us. URL: https://www.ibm.com/topics/industry-4-0 (visited on 02/28/2024).

[31] Melisa López et al. "Towards the 5G-Enabled Factories of the Future". In: *2023 IEEE 21st International Conference on Industrial Informatics (INDIN)*. ISSN: 2378-363X. July 2023, pp. 1–8. DOI: 10.1109/INDIN51400.2023.10217837. URL: https://ieeexplore.ieee.org/document/10217837 (visited on 07/17/2024).

[32] Gateworks. *Newport GW6400 Single Board Computer*. en-US. URL: https://www.gateworks.com/products/industrial-single-board-computers/octeon-tx-single-board-computers-gateworks-newport/gw6400-single-board-computer/ (visited on 07/17/2024).

[33] SimCom. *SIM8262X-M2 Series*. URL: https://www.simcom.com/product/SIM8262X-M2.html (visited on 07/17/2024).

[34] *Our history*. en. URL: https://www.universal-robots.com/about-universal-robots/our-history/,%20https://www.universal-robots.com/about-universal-robots/our-history/ (visited on 07/13/2024).

[35] *FORSIDE_release | The Gripper Company*. en-US. URL: https://thegrippercompany.com/forside_release/ (visited on 07/14/2024).

[36] *Real-Time Data Exchange (RTDE) Guide - 22229*. URL: https://www.universal-robots.com/articles/ur/interface-communication/real-time-data-exchange-rtde-guide/ (visited on 07/14/2024).

[37] *MiR100 - Cost-Effective Mobile Robot*. en-US. URL: https://www.industrialcontrol.com/mir100 (visited on 07/16/2024).

[38] *MiR 100 robot móvil | Vinssa*. es. URL: https://www.vinssa.com/mobile-industrial-robots/mir-100/ (visited on 07/16/2024).

[39] Robotics Alias. *Teardown - Mobile Industrial Robots MiR100*. URL: https://aliasrobotics.com/teardown-mir100.php (visited on 07/16/2024).

[40] *About*. en. URL: https://www.enabled-robotics.com/about (visited on 07/14/2024).

[41] *Software*. en. URL: https://www.enabled-robotics.com/software (visited on 07/14/2024).

[42] Barbara Simons. "An overview of clock synchronization". en. In: *Fault-Tolerant Distributed Computing*. Ed. by Barbara Simons and Alfred Spector. New York, NY: Springer, 1990, pp. 84–96. ISBN: 978-0-387-34812-4. DOI: 10.1007/BFb0042327.

[43] Markus Ullmann and Matthias Vögeler. "Delay attacks — Implication on NTP and PTP time synchronization". In: *Control and Communication 2009 International Symposium on Precision Clock Synchronization for Measurement*. ISSN: 1949-0313. Oct. 2009, pp. 1–6. DOI: 10.1109/ISPCS.2009.5340224. URL: https://ieeexplore.ieee.org/document/5340224 (visited on 07/16/2024).

[44] Liang Gong et al. "preciseSLAM: Robust, Real-Time, LiDAR–Inertial–Ultrasonic Tightly-Coupled SLAM With Ultraprecise Positioning for Plant Factories". In: *IEEE Transactions on Industrial Informatics* 20.6 (June 2024). Conference Name: IEEE Transactions on Industrial Informatics, pp. 8818–8827. ISSN: 1941-0050. DOI: 10.1109/TII.2024.3361092. URL: https://ieeexplore.ieee.org/document/10480585 (visited on 07/16/2024).

[45] P. Arpesi et al. "Rubidium Pulsed Optically Pumped Clock for Space Industry". In: *2019 Joint Conference of the IEEE International Frequency Control Symposium and European Frequency and Time Forum (EFTF/IFC)*. ISSN: 2327-1949. Apr. 2019, pp. 1–3. DOI: 10.1109/FCS.2019.8856140. URL: https://ieeexplore.ieee.org/document/8856140 (visited on 07/16/2024).

[46] R. Bridgham, G. Winkler, and F.H. Reder. “Synchronized Clock Experiment”. In: *13th Annual Symposium on Frequency Control*. May 1959, pp. 342–349. DOI: 10.1109/FREQ.1959.199390. URL: https://ieeexplore.ieee.org/document/1536330 (visited on 07/16/2024).

[47] Hermann Kopetz and Wilhelm Ochsenreiter. “Clock Synchronization in Distributed Real-Time Systems”. In: *IEEE Transactions on Computers* C-36.8 (Aug. 1987). Conference Name: IEEE Transactions on Computers, pp. 933–940. ISSN: 1557-9956. DOI: 10.1109/TC.1987.5009516. URL: https://ieeexplore.ieee.org/abstract/document/5009516 (visited on 07/16/2024).

[48] Baoying Wei, Kun Liang, and Tian Yu. “Simulation and Evaluation of Time Synchronization Performance Based on NTP and PTP”. In: *2023 Joint Conference of the European Frequency and Time Forum and IEEE International Frequency Control Symposium (EFTF/IFCS)*. ISSN: 2327-1949. May 2023, pp. 1–4. DOI: 10.1109/EFTF/IFCS57587.2023.10272125. URL: https://ieeexplore.ieee.org/document/10272125 (visited on 07/16/2024).

[49] *How NTP Works*. en. Section: documentation. URL: https://www.ntp.org/documentation/4.2.8-series/warp/ (visited on 07/05/2024).

[50] Faten Mkacher and Andrzej Duda. “Calibrating NTP”. In: *2019 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control, and Communication (ISPCS)*. ISSN: 1949-0313. Sept. 2019, pp. 1–6. DOI: 10.1109/ISPCS.2019.8886646. URL: https://ieeexplore.ieee.org/document/8886646 (visited on 07/16/2024).

[51] geeks. *Network Time Protocol (NTP) - GeeksforGeeks*. URL: https://www.geeksforgeeks.org/network-time-protocol-ntp/ (visited on 07/16/2024).

[52] chrony. *chrony – chrony.conf(5)*. URL: https://chrony-project.org/doc/4.4/chrony.conf.html (visited on 07/08/2024).

[53] Pedro Moreira et al. “White rabbit: Sub-nanosecond timing distribution over ethernet”. In: *Control and Communication 2009 International Symposium on Precision Clock Synchronization for Measurement*. ISSN: 1949-0313. Oct. 2009,

pp. 1–5. DOI: 10.1109/ISPCS.2009.5340196. URL: https://ieeexplore.ieee.org/abstract/document/5340196 (visited on 07/13/2024).

[54] Paola Iovanna et al. "SDN-based architecture to support Synchronization in a 5G framework". In: *2016 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control, and Communication (ISPCS)*. ISSN: 1949-0313. Sept. 2016, pp. 1–4. DOI: 10.1109/ISPCS.2016.7579504. URL: https://ieeexplore.ieee.org/document/7579504 (visited on 07/16/2024).

[55] Eoin O'Connell, Denis Moore, and Thomas Newe. "Challenges Associated with Implementing 5G in Manufacturing". In: *Telecom* 1 (June 2020), pp. 48–67. DOI: 10.3390/telecom1010005.

[56] Nima Enayati et al. "Using Cellular Connectivity for On-the-move Cooperation of Stationary Manipulator and Mobile Platform". In: *2021 26th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*. Sept. 2021, pp. 1–8. DOI: 10.1109/ETFA45728.2021.9613325. URL: https://ieeexplore.ieee.org/document/9613325 (visited on 07/13/2024).

[57] K. A. Venkataraman, K. Kanthavel, and B. Nirmal Kumar. "Investigations of Response Time Parameters of a Pneumatic 3/2 Direct Acting Solenoid Valve Under Various Working Pressure Conditions". en. In: *Engineering, Technology & Applied Science Research* 3.4 (Aug. 2013). Number: 4, pp. 502–505. ISSN: 1792-8036. DOI: 10.48084/etasr.360. URL: https://etasr.com/index.php/ETASR/article/view/360 (visited on 07/18/2024).

[58] blender. *Index of /peach/bigbuckbunny_movies/*. URL: https://download.blender.org/peach/bigbuckbunny_movies/ (visited on 07/17/2024).

A. Code developed for the Use Case Implementation

Inside this annex, all the code developed for the industrial use case is gathered.

A.1.- mqtt_Slave_static.py

This is the Python code for the Slave controller at the static robot ("mqtt_Slave_static.py").

```python
import sys
sys.path.append("..")
import threading
import time
import socket
import logging
import json
from collections import namedtuple
import paho.mqtt.client as mqtt
# Using Universal Robots RTDE libraries
import rtde.rtde as rtde
import rtde.rtde_config as rtde_config

import consts
import my_tools


# =========================================== RTDE CONFIGURATION
    =====================================================

# We start the motion variable to an initial position value,
# before getting data from RTDE we need to have something
motion = [0, -1.57, -1.57, -1.57, 1.57, 0]
# Set up a IP and port of the RTDE server inside the robot
```

```python
ROBOT_HOST = "10.42.0.233"
ROBOT_PORT = 30004
# Configuration file to follow
config_filename = "control_loop_configuration.xml"

# We apply the configuration file of RTDE
conf = rtde_config.ConfigFile(config_filename)
state_names, state_types = conf.get_recipe("state")
actualq_names, actualq_types = conf.get_recipe("actualq")
sdo_names, sdo_types = conf.get_recipe("sdo")  # standard digital output
speed_names, speed_types = conf.get_recipe("speed")  # speed

# We start a RTDE connection
con = rtde.RTDE(ROBOT_HOST, ROBOT_PORT)
con.connect()
print("Connected to RTDE")

# get controller version
con.get_controller_version()

# setup recipes
con.send_output_setup(state_names, state_types)
actual_q = con.send_input_setup(actualq_names, actualq_types)
sdo = con.send_input_setup(sdo_names, sdo_types)  # standard digital output
speed = con.send_input_setup(speed_names, speed_types)  # speed

# We apply the initial position to the joints
actual_q.input_double_register_24 = motion[0]
actual_q.input_double_register_25 = motion[1]
actual_q.input_double_register_26 = motion[2]
actual_q.input_double_register_27 = motion[3]
actual_q.input_double_register_28 = motion[4]
actual_q.input_double_register_29 = motion[5]
# start data synchronization
if not con.send_start():
    sys.exit()

```

```python
# We send to the robot via RTDE the initial position
con.send(actual_q)

# We configure Digital output
# standard digital output
sdo.standard_digital_output_mask = 255  # This is 7 "ones" in the binary
    system, which means all 7 digits are 1. Without this, we cannot change
    standard digital outputs
sdo.standard_digital_output = 0

# The arms cannot move too fast, or else the bases of them will shake and be
     not stable. Therefore every time when we run the program, we set the
    speed to 50%.
speed.speed_slider_mask = 4294967295  # This is 32 "ones" in the binary
    system, which means all 32 digits are 1.
speed.speed_slider_fraction = 0.2

# start data synchronization
if not con.send_start():
    sys.exit()

con.send(sdo)  # standard digital output
con.send(speed)  # speed

# ============================================ RTDE CONFIGURATION
    ======================================================

# Topics for MQTT
#print("Please input the index of the arm, starting from 1, 2, 3, ...")
arm_idx = 1
topic_sub = "{}{}".format(consts.TOPIC_TGT_PREFIX, arm_idx)
topic_pub = "{}{}".format(consts.TOPIC_CUR_PREFIX, arm_idx)
topic_grip = "{}{}".format(consts.TOPIC_GRIP_PREFIX, arm_idx)

# Current position of the arm
current_RTDE = None

```

```python
# to notify the "publish" thread that the "rtde" thread receives a new
    current position
e_new_cur = threading.Event()

# The timestamp when lasted message was received
last_recv_time = time.time()


# Function used to read the data of the joints using RTDE
def read_RTDE():
    global current_RTDE
    conf = rtde_config.ConfigFile("record_configuration.xml")
    output_names, output_types = conf.get_recipe("out")
    # get controller version
    con.get_controller_version()

    # setup recipes
    if not con.send_output_setup(
            output_names, output_types, frequency=consts.RTDE_FREQ):
        logging.error("Unable to configure output")
        sys.exit()

    # start data synchronization
    if not con.send_start():
        logging.error("Unable to start synchronization")
        sys.exit()

    keep_running = True
    while keep_running:
        try:
            # Getting data from the joints from RTDE
            # This will block if there is not new data.
            state = con.receive()
            # We convert the data into json object
            data = json.dumps(state.__dict__)  # serialize data object
            # Converting JSON data into Python readable object
            current_RTDE = json.loads(
                data,
                object_hook=lambda d: namedtuple('a', d.keys())(*d.values()))
            # notify another thread to report this current position
            e_new_cur.set()
        except KeyboardInterrupt:
            keep_running = False
        except rtde.RTDEException:
            con.disconnect()
            sys.exit()

    con.send_pause()
    con.disconnect()


# Function used to write the joint with the values we get from the edge cloud
def write_RTDE(motion):

    actual_q.input_double_register_24 = motion[0]
    actual_q.input_double_register_25 = motion[1]
    actual_q.input_double_register_26 = motion[2]
    actual_q.input_double_register_27 = motion[3]
    actual_q.input_double_register_28 = motion[4]
    actual_q.input_double_register_29 = motion[5]

    con.send(actual_q)
    print("POSITION HAS APPLIED TO THE ROBOT USING RTDE")
    print(motion)


def on_connect(client, userdata, flags, return_code):
    if return_code == 0:
        print("UR Robot connected to broker!")
        # We subscribe
        client.subscribe(topic_sub)
        print("Subscribed to topic: " + topic_sub)
        client.subscribe(topic_grip)
        print("Subscribed to topic: " + topic_grip)
        print("WE HAVE SUBSCRIBED TO THE BROKER")
    else:
        print("could not connect to mqtt broker, return code:", return_code)


def on_message(client, userdata, message):
    global motion, last_recv_time
    try:
        # We receive the message of the motion of the other robot
        topic = message.topic
        if topic == topic_sub:
            # We need to measure "Uplink time", "Downlink time", and "Processing time", but the "gripper" messages do not nave uplink time, so we only research the time of "Arm movements" messages.
            last_recv_time = time.time()

            message_float = message.payload.decode("utf-8")
            message = eval(message_float)
            message_motion = message[:-1]
            goals_time = message[-1]
            while True:
                current_time = time.time()
                if current_time > goals_time:
                    motion = message_motion
                    write_RTDE(motion)
                    break
                time.sleep(0.001)
        elif topic == topic_grip:
            grip_recv_time = time.time()
            message_float2 = message.payload.decode("utf-8")
            message2 = eval(message_float2)
            message_grip = message2[:-1]
            goals_time2 = message2[-1]
            while True:
                current_time = time.time()
                if current_time > goals_time2:
                    # change_sdo(message_grip[0]) # This was aimed to become an intermediate state, but this design seems unnecessary and this implementation is too simple, so I remove this unnecessary behavior.
                    change_sdo(message_grip[1])
                    if (message_grip[1] == 1):
                        #current_time = time.time()
                        my_tools.publish(client, "demo_gripper", "[{}, {}]".format(2, grip_recv_time))
                    break
                time.sleep(0.001)
        else:
            print("Message from unknown topic: {}".format(topic))

    # We convert the string list to a normal list
    except Exception as e:
        print("EXCEPTION: ", e)


# change standard digital outputs.
def change_sdo(value: int):
    sdo.standard_digital_output = value
    con.send(sdo)


# report the current joints position
def report_cur_pos():
    # We create an old list to know the old position
    old_current_pos_list = ""
    while True:

        # if rtde does not get a new current position, the code will be blocked here.
        # Here, we do not need a timeout, although the possibility that "a set" of the event is before clear() after reading the shared_value still exists, the read_RTDE thread calls e_new_cur.set() every RTDE period, so there is always the next set after a set. However, if we have a timeout,
    at the beginning of this program, if read_RTDE thread thread does not set a value before the timeout here, the following code will crash.
        e_new_cur.wait()

        # With the above e_new_cur.wait, we do not need this check None, because only after rtde get set a value to current_RTDE, rtde will do e_new_cur.set(), unless rtde gives us a None value.
        # if current_RTDE is None:
        #     # at the beginning this variable is None, which will make the following code crash, so we should continue here
        #     continue

        tmp_current_RTDE = current_RTDE  # in case of current_RTDE changes during the following code

        # clear the event after reading the latest current positions, so the next loop will still be blocked until rtde receives the next current position
        e_new_cur.clear()

        # We store the current position
        current_pos_list = "[" + str(tmp_current_RTDE.actual_q[0]) + "," + str(
            tmp_current_RTDE.actual_q[1]) + "," + str(
                tmp_current_RTDE.actual_q[2]) + "," + str(
                    tmp_current_RTDE.actual_q[3]) + "," + str(
                        tmp_current_RTDE.actual_q[4]) + "," + str(
                            tmp_current_RTDE.actual_q[5]) + "]"
        # If our last position (old) and the current one are different, we publish our last position
        if len(old_current_pos_list) == 0 or (not my_tools.joints_pos_equal(
                eval(old_current_pos_list), eval(current_pos_list))):
            cur_send_time = time.time()
            cur_pos_with_times = "[{}, {}, {}]".format(current_pos_list,
                                                        last_recv_time,
                                                        cur_send_time)
            # publish current positions, downlink message received time, and
    uplink message sent time
            my_tools.publish(client, topic_pub, cur_pos_with_times)
            old_current_pos_list = current_pos_list


client = mqtt.Client()
client.username_pw_set(username=consts.BROKER_USER,
                       password=consts.BROKER_PASSWD)
client.on_connect = on_connect  # subscribe is in this on_connect
client.on_message = on_message
client.connect(consts.BROKER_IP,
               consts.BROKER_PORT)  # here, the subscribe will be executed
client.socket().setsockopt(socket.IPPROTO_TCP, socket.TCP_NODELAY, True)
client.loop_start()

# We start a thread used for reading the joint values using RTDE
thread_read = threading.Thread(target=read_RTDE)
thread_read.start()

try:
    # report the current joints position
    report_cur_pos()
finally:
    client.loop_stop()
    client.unsubscribe(topic_sub)
    client.disconnect()
```

A.2.- mqtt_Slave_mobile.py

This is the Python code for the Slave controller at the mobile robot ("mqtt_Slave_mobile.py").

```python
import sys

sys.path.append("..")

import logging
import json

import paho.mqtt.client as mqtt

# Using Universal Robots RTDE libraries
import rtde.rtde as rtde
import rtde.rtde_config as rtde_config

from collections import namedtuple
import threading
import time
import socket

import consts
import my_tools
import mir_api_s

import requests
import urllib3

# We start the motion variable to none
# before getting data from RTDE we need to have something even though it is
    empty
# Initial position
motion = [0.5, -2.007, -2.269, -2.007, -1.57, 1.57]
demo_state = "Start"
demo_mision = "None"
demo_gripper = 0
```

```
topic_mir_control = "mir_control"
topic_demo_ready = "demo_ready"
topic_demo_state = "demo_state"
topic_demo_mision = "mir_control"#"demo_mision"
topic_demo_gripper = "demo_gripper"
ready_sent = False
mir_csv_data = []

# ============================================ RTDE CONFIGURATION
    ======================================================

# Set up a RTDE server inside the robot
# Set the IP
ROBOT_HOST = "192.168.12.40"
# Set the port
ROBOT_PORT = 30004
# Config to follow
config_filename = "control_loop_configuration.xml"

# logging.getLogger().setLevel(logging.INFO)

# We apply the configuration file of RTDE
conf = rtde_config.ConfigFile(config_filename)
state_names, state_types = conf.get_recipe("state")
actualq_names, actualq_types = conf.get_recipe("actualq")
# watchdog_names, watchdog_types = conf.get_recipe("watchdog")

#sdo_names, sdo_types = conf.get_recipe("sdo")  # standard digital output
#speed_names, speed_types = conf.get_recipe("speed")  # speed

# We start a RTDE connection
con = rtde.RTDE(ROBOT_HOST, ROBOT_PORT)
con.connect()
print("Connected to RTDE")

# get controller version
con.get_controller_version()
```

```

# setup recipes
con.send_output_setup(state_names, state_types)
actual_q = con.send_input_setup(actualq_names, actualq_types)
#sdo = con.send_input_setup(sdo_names, sdo_types)  # standard digital output
#speed = con.send_input_setup(speed_names, speed_types)  # speed

# We apply the initial position to the joints
actual_q.input_double_register_24 = motion[0]
actual_q.input_double_register_25 = motion[1]
actual_q.input_double_register_26 = motion[2]
actual_q.input_double_register_27 = motion[3]
actual_q.input_double_register_28 = motion[4]
actual_q.input_double_register_29 = motion[5]
# start data synchronization
if not con.send_start():
    sys.exit()

# We send to the robot via RTDE the initial position
con.send(actual_q)

# We configure Digital output
# standard digital output
#sdo.standard_digital_output_mask = 255  # This is 7 "ones" in the binary
    system, which means all 7 digits are 1. Without this, we cannot change
    standard digital outputs
#sdo.standard_digital_output = 0

# The arms cannot move too fast, or else the bases of them will shake and be
     not stable. Therefore every time when we run the program, we set the
    speed to 50%.
#speed.speed_slider_mask = 4294967295  # This is 32 "ones" in the binary
    system, which means all 32 digits are 1.
#speed.speed_slider_fraction = 0.5

# start data synchronization
if not con.send_start():
    sys.exit()

#con.send(sdo)  # standard digital output
#con.send(speed)  # speed

# ============================================ RTDE CONFIGURATION
    ======================================================

# Topics for MQTT
#print("Please input the index of the arm, starting from 1, 2, 3, ...")
arm_idx = 2
topic_sub = "{}{}".format(consts.TOPIC_TGT_PREFIX, arm_idx)
topic_pub = "{}{}".format(consts.TOPIC_CUR_PREFIX, arm_idx)
topic_pub2 = "{}{}".format(consts.TOPIC_RDY_PREFIX, arm_idx)
topic_grip = "{}{}".format(consts.TOPIC_GRIP_PREFIX, arm_idx)

# Current position of the arm
current_RTDE = None

# to notify the "publish" thread that the "rtde" thread receives a new
    current position
e_new_cur = threading.Event()

# The timestamp when lasted message was received
last_rx_arm = time.time()
last_tx_grp = time.time()
last_rx_mir = time.time()
last_tx_mir = time.time()


# Function used to read the data of the joints using RTDE
def read_RTDE():
    global current_RTDE
    conf = rtde_config.ConfigFile("record_configuration.xml")
    output_names, output_types = conf.get_recipe("out")
    # get controller version
    con.get_controller_version()
```

```python

    # setup recipes
    if not con.send_output_setup(
            output_names, output_types, frequency=consts.RTDE_FREQ):
        logging.error("Unable to configure output")
        sys.exit()

    # start data synchronization
    if not con.send_start():
        logging.error("Unable to start synchronization")
        sys.exit()

    keep_running = True
    while keep_running:
        try:
            # Getting data from the joints from RTDE
            # This will block if there is not new data.
            state = con.receive()
            # We convert the data into json object
            data = json.dumps(state.__dict__)  # serialize data object
            # Converting JSON data into Python readable object
            current_RTDE = json.loads(
                data,
                object_hook=lambda d: namedtuple('a', d.keys())(*d.values()))
            # notify another thread to report this current position
            e_new_cur.set()
        except KeyboardInterrupt:
            keep_running = False
        except rtde.RTDEException:
            con.disconnect()
            sys.exit()

    con.send_pause()
    con.disconnect()


```

```python
# Function used to write the joint with the values we get from the edge
    cloud
def write_RTDE(motion):

    actual_q.input_double_register_24 = motion[0]
    actual_q.input_double_register_25 = motion[1]
    actual_q.input_double_register_26 = motion[2]
    actual_q.input_double_register_27 = motion[3]
    actual_q.input_double_register_28 = motion[4]
    actual_q.input_double_register_29 = motion[5]

    con.send(actual_q)
    print("POSITION HAS APPLIED TO THE ROBOT USING RTDE")
    print(motion)


def on_connect(client, userdata, flags, return_code):
    if return_code == 0:
        print("UR Robot connected to broker!")
        # We subscribe
        client.subscribe(topic_sub)
        print("Subscribed to topic: " + topic_sub)
        client.subscribe(topic_grip)
        print("Subscribed to topic: " + topic_grip)
        client.subscribe(topic_demo_mision)
        print("Subscribed to topic: " + topic_demo_mision)
        print("WE HAVE SUBSCRIBED TO THE BROKER")
    else:
        print("could not connect to mqtt broker, return code:", return_code)


def on_message(client, userdata, message):
    global motion, last_rx_arm, last_rx_grp, last_rx_mir, demo_state,
   demo_mision, ready_sent
    try:
        # We receive the message of the motion of the other robot
        topic = message.topic
        if topic == topic_sub:
            # We need to measure "Uplink time", "Downlink time", and "Processing time", but the "gripper" messages do not nave uplink time, so we only research the time of "Arm movements" messages.
            last_rx_arm = time.time()

            message_float = message.payload.decode("utf-8")
            message = eval(message_float)
            message_motion = message[:-1]
            goals_time = message[-1]
            while True:
                current_time = time.time()
                if current_time > goals_time:
                    motion = message_motion
                    write_RTDE(motion)
                    ready_sent = False
                    break
                time.sleep(0.001)
        # No need to evaluate gripper -> There's no gripper on ER robot
        # We receive the demo mission, no need for the demo state
        #elif topic == topic_demo_state:
        #    demo_state = message.payload.decode("utf-8")
        elif topic == topic_demo_mision:
            last_rx_mir = time.time()
            demo_mision = message.payload.decode("utf-8")

        elif topic == topic_grip:
            last_rx_grp = time.time()

        else:
            print("Message from unknown topic: {}".format(topic))

    # We convert the string list to a normal list
    except Exception as e:
        print("EXCEPTION: ", e)


```

```
# change standard digital outputs.
#def change_sdo(value: int):
#     sdo.standard_digital_output = value
#     con.send(sdo)


# report the current joints position
def report_cur_pos():
    global ready_sent
    # We create an old list to know the old position
    old_current_pos_list = ""
    while True:

        # if rtde does not get a new current position, the code will be blocked here.
        # Here, we do not need a timeout, although the possibility that "a set" of the event is before clear() after reading the shared_value still exists, the read_RTDE thread calls e_new_cur.set() every RTDE period, so there is always the next set after a set. However, if we have a timeout, at the beginning of this program, if read_RTDE thread thread does not set a value before the timeout here, the following code will crash.
        e_new_cur.wait()

        # With the above e_new_cur.wait, we do not need this check None, because only after rtde get set a value to current_RTDE, rtde will do e_new_cur.set(), unless rtde gives us a None value.
        # if current_RTDE is None:
        #     # at the beginning this variable is None, which will make the following code crash, so we should continue here
        #     continue

        tmp_current_RTDE = current_RTDE  # in case of current_RTDE changes during the following code

        # clear the event after reading the latest current positions, so the next loop will still be blocked until rtde receives the next current position
        e_new_cur.clear()

        # We store the current position
        current_pos_list = "[" + str(tmp_current_RTDE.actual_q[0]) + "," + str(
            tmp_current_RTDE.actual_q[1]) + "," + str(
                tmp_current_RTDE.actual_q[2]) + "," + str(
                    tmp_current_RTDE.actual_q[3]) + "," + str(
                        tmp_current_RTDE.actual_q[4]) + "," + str(
                            tmp_current_RTDE.actual_q[5]) + "]"
        # If our last position (old) and the current one are different, we publish our last position
        if len(old_current_pos_list) == 0 or (not my_tools.joints_pos_equal(
                eval(old_current_pos_list), eval(current_pos_list))):
            cur_send_time = time.time()
            cur_pos_with_times = "[{}, {}, {}]".format(current_pos_list,
                                                       last_rx_arm,
                                                       cur_send_time)
            # publish current positions, downlink message received time, and uplink message sent time
            my_tools.publish(client, topic_pub, cur_pos_with_times)
            old_current_pos_list = current_pos_list
        else:
            my_tools.publish(client, topic_pub2, cur_pos_with_times)

def state_MiR(mir):
    global demo_state, demo_mision, demo_gripper, ready_sent, mir_csv_data
    t1 = time.time_ns()
    response_json = mir.get_system_info(mir.local_link)
    t2 = time.time_ns()
    req_rtt_ms = ((t2-t1)/1000000)
    state = [response_json["state_text"], response_json["velocity"]["linear"], response_json["velocity"]["angular"],
             response_json["position"]["x"], response_json["position"]["y"], response_json["position"]["orientation"],
             req_rtt_ms, demo_state, demo_mision, demo_gripper, ready_sent, t1]
    print("Last response: ",req_rtt_ms)
    mir_csv_data.append(state)
    return state

def control_MiR():
    global demo_state, demo_mision, demo_gripper, ready_sent, mir_csv_data

    mir = mir_api_s.MiR()
    prev_demo_mision = demo_mision
    keep_running_cMiR = True
    while keep_running_cMiR:
        time.sleep(0.01)
        try:
            if (prev_demo_mision != demo_mision):
                print("-------------Post mission------------")
                result = mir.post_to_mission_queue(mir.local_link, mir.
mission_dict[demo_mision])
                prev_demo_mision = demo_mision
                demo_state = "Not Ready"
                time.sleep(1)
            state = state_MiR(mir)
            if (((state[0] == "Ready" and state[1] == 0)) and demo_state!="
Ready"):
                demo_state = "Ready"
                my_tools.publish(client, topic_demo_ready, demo_state)
                if (demo_gripper == 1):
                    ready_sent = False
                    demo_gripper = 0
                    my_tools.write_csv_file("saved_data/data_mir_" + str(int
(time.time())) + ".csv", mir_csv_data)
                    mir_csv_data.clear()
            if (((abs(state[3] - 4.15 +0.07)<0.07)) and demo_gripper!=1 and
demo_state!="Ready"):
                demo_gripper = 1
                current_time = time.time()
                my_tools.publish(client, topic_demo_gripper, "[{}, {}]".
format(demo_gripper, current_time))
```

```

        except KeyboardInterrupt:
            keep_running_cMiR = False
        except:
            print("No response obtained")




client = mqtt.Client()
client.username_pw_set(username=consts.BROKER_USER,
                       password=consts.BROKER_PASSWD)
client.on_connect = on_connect  # subscribe is in this on_connect
client.on_message = on_message
client.connect(consts.BROKER_IP,
               consts.BROKER_PORT)  # here, the subscribe will be executed
client.socket().setsockopt(socket.IPPROTO_TCP, socket.TCP_NODELAY, True)
client.loop_start()

# We start a thread used for reading the joint values using RTDE
thread_read = threading.Thread(target=read_RTDE)
thread_read.start()

# We start a thread used for moving the MiR robot
thread_mir = threading.Thread(target=control_MiR)
thread_mir.start()

try:
    # report the current joints position
    report_cur_pos()
finally:
    client.loop_stop()
    client.unsubscribe(topic_sub)
    client.disconnect()
```

A.3.- mqtt_Slave_controler.py

This is the Python code for the Master controller at the Edge-Cloud server ("mqtt_Master_controler.py").

```python
import math
from paho.mqtt import client as mqtt_client
import time
import logging
import threading
import socket

import consts
import mir_api_s
import my_tools

# ---- This controller is made to rx the current position of the slave 2 and
    tx two fixed positions ----

# For measurement, to record the key time points of a round of messages.
class OneRoundTimes:
    def __init__(self):
        # This class is for measurement, in every round, we should record:
        # the time when controller sends the target position
        self.t_c_send_tgt = time.time()
        # the time when slave receives the target position
        self.t_s_recv_tgt = time.time()
        # the time when slave sends the current position
        self.t_s_send_cur = time.time()
        # the time when controller receives the current position
        self.t_c_recv_cur = time.time()

times1 = OneRoundTimes()
times2 = OneRoundTimes()
timesg = OneRoundTimes()
timesmir = OneRoundTimes()
```

```python
timesg_ = []

arm1_csv_data = []
arm2_csv_data = []
armg_csv_data = []
armmir_csv_data = []

# Current position (subscribed)
current_pos_with_time_1 = None
current_pos_with_time_2 = None

# Number of iterations
pair = 0
demo_state = "None"
demo_mision = "None"
demo_counter = 0
demo_next_ready = "True"
demo_gripper = "Hold"

topic_mir_control = "mir_control"
topic_mir_state = "mir_state"
topic_demo_state = "demo_state"
topic_demo_ready = "demo_ready"
topic_demo_gripper = "demo_gripper"

demo_dict = {
    0: "None",
    1: "Start",
    2: "Initial",
    3: "Positioned",
    4: "Placed",
    5: "Loaded",
    6: "Finishing",
    7: "End"
}

def deg_to_rad(motiond_l):
    motionr_l=[0,0,0,0,0,0]
    for i in range(len(motiond_l)):
        motionr_l[i]=motiond_l[i]*math.pi/180
    return motionr_l

static_arm0_deg=[0,-92.5,-92.5,-80,90,0]
static_arm1_deg=[-90,-92.5,-92.5,-80,90,0]
motion_deg1=[30,-115,-130,-115,-90,90]
motion_deg2=[-90,-115,-130,-115,-90,90]
#motion_deg3=[-50,-125,-110,-125,-90,90]
static_arm0_rad=deg_to_rad(static_arm0_deg)
static_arm1_rad=deg_to_rad(static_arm1_deg)
motion_rad1=deg_to_rad(motion_deg1)
motion_rad2=deg_to_rad(motion_deg2)
#motion_rad3=deg_to_rad(motion_deg3)

# to notify the "publish" thread that the "subscribe" thread receives a new
    current position
e_new_cur = threading.Event()

# The time to wait until executing the behaviors
#print("Please input the timestamp_to_delay in second")
timestamp_to_delay = 0.02 #float(input())


# Function for connecting the broker
def connect_mqtt() -> mqtt_client.Client:
    def on_connect(client, userdata, flags, rc):
        if rc == 0:
            print("Robot operator has connected to MQTT Broker!\n")
        else:
            print("Failed to connect, return code %d\n", rc)

    client = mqtt_client.Client()
    # Uncomment when using authenticated communications
    client.username_pw_set(username="testbed", password="1234")
    client.on_connect = on_connect
```

```python
    client.connect(consts.BROKER_IP, consts.BROKER_PORT)
    client.socket().setsockopt(socket.IPPROTO_TCP, socket.TCP_NODELAY, True)
    return client

# SUBSCRIBER function
def subscribe(client: mqtt_client.Client):

    def on_message(client, userdata, msg):
        global demo_state, demo_counter, demo_gripper, demo_next_ready,
   current_pos_with_time_1, current_pos_with_time_2, timesg_, armg_csv_data
        recv_time = time.time()
        # Message received
        topic = msg.topic
        rx_msg = msg.payload.decode() #"utf-8"
        if msg.topic == topic_demo_state:
            demo_state = rx_msg
        elif msg.topic == topic_demo_ready:
            demo_ready = rx_msg
            if (demo_ready == "Ready"):
                print("----Next step----")
                demo_counter = demo_counter+1
                demo_state = demo_dict[demo_counter]
                demo_next_ready = True
        elif msg.topic == topic_demo_gripper:
            demo_gripper = eval(rx_msg)[0]
            print("Msg: demo_gripper", demo_gripper)
            if (demo_gripper == 1):
                print("----Next step----")
                timesg.t_c_send_tgt = eval(rx_msg)[1]
                timesg.t_s_recv_tgt = recv_time
                timesg_.append(eval(rx_msg)[1])
                print(eval(rx_msg)[1])
                timesg_.append(recv_time)
                demo_counter = demo_counter+1
                demo_state = demo_dict[demo_counter]
                demo_next_ready = True
            elif (demo_gripper == 2):
                print("----Gripper execution----")
                timesg.t_c_recv_cur = eval(rx_msg)[1]
                timesg_.append(eval(rx_msg)[1])
                print(eval(rx_msg)[1])
                armg_csv_data.append(timesg_)
                #armg_csv_data.append([
                #    timesg.t_c_send_tgt, timesg.t_s_recv_tgt, timesg.
   t_s_send_cur, timesg.t_c_recv_cur])


        else:
            # the original message has 3 elements:
            # 1. current positions;
            # 2. target position received time;
            # 3. current position sent time.
            # We append the current position received time to it as the 4th
   element.
            tmp = eval(rx_msg)
            tmp.append(recv_time)

            if topic == consts.TOPIC_CUR1:
                current_pos_with_time_1 = tmp
            elif topic == consts.TOPIC_CUR2:
                current_pos_with_time_2 = tmp
            else:
                #print()"Message from unknown topic: {}, this \"on_message\"
    will return."format(topic))
                return

            e_new_cur.set(
            )  # notify the "wait_pos" function that we receives a new
   current position, and "wait_pos" will check that position.
            """
            print("---------------------------------")
            print("Received message " + rx_msg)
            print("Topic: " + rx_topic)
            print("---------------------------------")
            """
    client.subscribe(consts.TOPIC_CUR1, qos=consts.QOS_LEVEL)
    print("Subscribed to topic: " + consts.TOPIC_CUR1)
    client.subscribe(consts.TOPIC_CUR2, qos=consts.QOS_LEVEL)
    print("Subscribed to topic: " + consts.TOPIC_CUR2)
    client.subscribe(consts.TOPIC_RDY1, qos=consts.QOS_LEVEL)
    print("Subscribed to topic: " + consts.TOPIC_RDY1)
    client.subscribe(consts.TOPIC_RDY2, qos=consts.QOS_LEVEL)
    print("Subscribed to topic: " + consts.TOPIC_RDY2)
    #client.subscribe(topic_mir_control, qos=consts.QOS_LEVEL)
    #print("Subscribed to topic: " + topic_mir_control)
    client.subscribe(topic_demo_state, qos=consts.QOS_LEVEL)
    print("Subscribed to topic: " + topic_demo_state)
    client.subscribe(topic_demo_ready, qos=consts.QOS_LEVEL)
    print("Subscribed to topic: " + topic_demo_ready)
    client.subscribe(topic_demo_gripper, qos=consts.QOS_LEVEL)
    print("Subscribed to topic: " + topic_demo_gripper)
    client.on_message = on_message

# loop_start will create a thread for subscribing
def subscriber(client: mqtt_client.Client):
    subscribe(client)
    client.loop_start()

# PUBLISH function
def publish(client: mqtt_client.Client, topic_pub, msg):
    # We publish the message
    result = client.publish(topic_pub, msg, qos=consts.QOS_LEVEL)
    status = result[0]
    if status == 0:
        pass
        # print(f"Successful! Sent message \"{msg}\" to topic {topic_pub}")
    else:
        print(f"Failed to send message \"{msg}\" to topic {topic_pub}")


def move_arm1(client, positions):
    current_time = time.time()
    do_time = current_time + timestamp_to_delay
    publish(
        client, consts.TOPIC_TGT1, "[{}, {}, {}, {}, {}, {}, {}]".format(
            positions[0], positions[1],
            positions[2], positions[3],
            positions[4], positions[5],
            do_time))
    # Instead of directly write them into files, we save them in variables
    and write to files later. This is to avoid the possible effects from I/O.
    arm1_csv_data.append([
            times1.t_c_send_tgt, times1.t_s_recv_tgt, times1.t_s_send_cur,
            times1.t_c_recv_cur
        ])

def move_arm2(client, positions):
    current_time = time.time()
    do_time = current_time + timestamp_to_delay
    publish(
        client, consts.TOPIC_TGT2, "[{}, {}, {}, {}, {}, {}, {}]".format(
            positions[0], positions[1],
            positions[2], positions[3],
            positions[4], positions[5],
            do_time))
    # Instead of directly write them into files, we save them in variables
    and write to files later. This is to avoid the possible effects from I/O.
    arm1_csv_data.append([
            times2.t_c_send_tgt, times2.t_s_recv_tgt, times2.t_s_send_cur,
            times2.t_c_recv_cur
        ])

def move_gripper(client, position):
    global timesg_, armg_csv_data
    current_time = time.time()
    do_time = current_time# + timestamp_to_delay
    publish(client, consts.TOPIC_GRIP1, "[{}, {}, {}]".format(3, position,
    do_time))
```

```python
    timesg.t_s_send_cur = current_time
    timesg_.append(current_time)

def move_mir(client, movement):
    global timesmir, armmir_csv_data
    current_time = time.time()
    do_time = current_time + timestamp_to_delay
    publish(client, topic_mir_control, movement)#"[{}, {}]".format(movement,
     do_time))
    armmir_csv_data.append([
            timesmir.t_c_send_tgt, timesmir.t_s_recv_tgt, timesmir.
    t_s_send_cur,
            timesmir.t_c_recv_cur
        ])

# Function used for executing the movement
def move_arms(client, pos_arm1, pos_arm2):
    global times1, times2, arm1_csv_data, arm2_csv_data

    e_new_cur.clear()  # unset the event
    current_time = time.time()
    do_time = current_time + timestamp_to_delay
    publish(
        client, consts.TOPIC_TGT1, "[{}, {}, {}, {}, {}, {}, {}]".format(
            pos_arm1[0], pos_arm1[1],
            pos_arm1[2], pos_arm1[3],
            pos_arm1[4], pos_arm1[5],
            do_time))
    # the above publish needs some time, because it includs network
    transmission, so we need to get the time again here, to accurately
    measure the data transmission time of the second arm.
    topic2_pub_time = time.time()
    publish(
        client, consts.TOPIC_TGT2, "[{}, {}, {}, {}, {}, {}, {}]".format(
            pos_arm2[0], pos_arm2[1],
            pos_arm2[2], pos_arm2[3],
            pos_arm2[4], pos_arm2[5],
            do_time))
    times1.t_c_send_tgt = current_time
    times2.t_c_send_tgt = topic2_pub_time
    wait_pos(pos_arm1, pos_arm2)
    # Instead of directly write them into files, we save them in variables
    and write to files later. This is to avoid the possible effects from I/O.
    arm1_csv_data.append([
            times1.t_c_send_tgt, times1.t_s_recv_tgt, times1.t_s_send_cur,
            times1.t_c_recv_cur
        ])
    arm2_csv_data.append([
            times2.t_c_send_tgt, times2.t_s_recv_tgt, times2.t_s_send_cur,
            times2.t_c_recv_cur
        ])

# Function used for waiting to the final position
def wait_pos(pos1, pos2):
    global times1, times2

    while True:
        # if controller does not receive a new current position from arms,
    the code will be blocked here.
        # The timeout is for the possibility that the that "the last set" of
     the event is before clear() after reading the shared_value. Therefore,
    we do a check if we have not received a new position in a double period
    of RTDE.
        e_new_cur.wait(timeout=1 / consts.RTDE_FREQ * 2)

        # Even we have the above e_new_cur.wait(), we still this "check None
    ", because e_new_cur will be set when either current_pos_with_time_1 or
    current_pos_with_time_2 is set, but the other one may still be None.
        if current_pos_with_time_2 is None:
            # at the beginning this variable is None, which will make the
    following code crash, so we should continue here
            continue

        # We get the current ur5 position of the joints and convert it to list
        # print("CURRENT POS B4 CONVERT: ", current_pos)
        dest_pos2 = pos2

        # in case of variables changes during the following code
        tmp_current_pos_with_time_2 = current_pos_with_time_2

        # clear the event after reading the latest current positions, so the next loop will still be blocked until receiving the next current position
        e_new_cur.clear()

        # The 1st element of current_pos_with_time_X is a stringed list of current joints positions
        current_mqtt_position2 = tmp_current_pos_with_time_2[0]

        time_tgt_recv_2 = tmp_current_pos_with_time_2[1]
        time_cur_sent_2 = tmp_current_pos_with_time_2[2]
        time_cur_recv_2 = tmp_current_pos_with_time_2[3]

        # both the 2 arms reaches their target positions
        if my_tools.joints_pos_equal(dest_pos2, current_mqtt_position2):
            break


def write_data():
    logging.info("Write measurement data into csv files.")
    #my_tools.write_csv_file(consts.DATA_FILE_ARM1, arm1_csv_data)
    #my_tools.write_csv_file(consts.DATA_FILE_ARM2, arm2_csv_data)
    my_tools.write_csv_file("saved_data/data_gripper_" + str(int(time.time())) + ".csv", armg_csv_data)
    #my_tools.write_csv_file(consts.DATA_FILE_ARM2+"m", armg_csv_data)
    #arm1_csv_data.clear()
    #arm2_csv_data.clear()
    armg_csv_data.clear()
    #armmir_csv_data.clear()
```

```python


# Main function
def run():
    global demo_state, demo_mision, demo_counter, demo_next_ready

    demo_counter = 0
    demo_state = demo_dict[demo_counter]
    demo_next_ready = True
    try:
        # We connect
        client = connect_mqtt()
        # subscribe and loop_start
        subscriber(client)

        mir = mir_api_s.MiR()

        while (demo_counter!=7):
            if (demo_counter == 0 and demo_next_ready == True):
                demo_next_ready = False
                print("Moving arm1, arm2 and gripper")
                g_position = 2 # Closed
                move_gripper(client, g_position)
                position_arm1 = static_arm0_rad
                position_arm2 = motion_rad1
                #move_arm1(client, position_arm1)
                #move_arm2(client, position_arm2)
                #wait_pos(position_arm1, position_arm2)
                move_arms(client, position_arm1, position_arm2)
                print("----Next step----") #Doesn't wait for the "Ready"
                demo_counter = demo_counter+1
                demo_state = demo_dict[demo_counter]
                demo_next_ready = True

            elif (demo_counter == 1 and demo_next_ready == True):
                demo_next_ready = False
                print("Moving mir to start of relative move")
                move_mir(client, "demo-relstart")
                #publish(client, topic_mir_control, "demo-relstart")#"[{}]".
format(position))

            elif (demo_counter == 2 and demo_next_ready == True):
                demo_next_ready = False
                print("Move both arms out")
                position_arm1 = static_arm1_rad
                position_arm2 = motion_rad2
                #move_arm1(client, position_arm1)
                #move_arm2(client, position_arm2)
                #wait_pos(position_arm1, position_arm2)
                move_arms(client, position_arm1, position_arm2)
                print("----Next step----") #Doesn't wait for the "Ready"
                demo_counter = demo_counter+1
                demo_state = demo_dict[demo_counter]
                demo_next_ready = True

            elif (demo_counter == 3 and demo_next_ready == True):
                demo_next_ready = False
                print("Start relative move")
                move_mir(client, "demo-relmove")
                #publish(client, topic_mir_control, "demo-relmove")#"[{}]".
format(position))
                #time.sleep(2+0.2) # 1/0.5

            elif (demo_counter == 4 and demo_next_ready == True):
                demo_next_ready = False
                print("Gripper opened")
                g_position = 1 # Opened
                move_gripper(client, g_position)

            elif (demo_counter == 5 and demo_next_ready == True):
                demo_next_ready = False
                print("Move tray back")
                g_position = 0 # Intermidiate
                move_gripper(client, g_position)
                position_arm1 = static_arm0_rad
                position_arm2 = motion_rad1
                #move_arm1(client, position_arm1)
                #move_arm2(client, position_arm2)
                #wait_pos(position_arm2)
                move_arms(client, position_arm1, position_arm2)
                print("----Next step----") #Doesn't wait for the "Ready"
                demo_counter = demo_counter+1
                demo_state = demo_dict[demo_counter]
                demo_next_ready = True
                write_data()

            elif (demo_counter == 6 and demo_next_ready == True):
                demo_next_ready = False
                print("Move MiR to finish position")
                move_mir(client, "demo-start")


            time.sleep(0.001)


    except (KeyboardInterrupt, SystemExit):
        print("KeyboardInterrupt")
    finally:
        client.loop_stop()
        client.unsubscribe(consts.TOPIC_CUR1)
        client.unsubscribe(consts.TOPIC_CUR2)
        client.disconnect()


if __name__ == '__main__':
    run()
```

A.4.- my_tools.py

This is the Python code with some of the used functions ("my_tools.py") that is used by both Slave programs.

```python
import csv
import paho.mqtt.client as mqtt

import consts


# PUBLISH function
def publish(client: mqtt.Client, topic_pub, msg):
    # We publish the message
    result = client.publish(topic_pub, msg, qos=consts.QOS_LEVEL)
    status = result[0]
    if status == 0:
        pass
        # print(f"Successful! Sent message \"{msg}\" to topic {topic_pub}")
    else:
        print(f"Failed to send message \"{msg}\" to topic {topic_pub}")


# Check whether 2 joints positions are equal
def joints_pos_equal(pos1: list[float], pos2: list[float]) -> bool:
    if len(pos1) != len(pos2):
        return False
    for i, _ in enumerate(pos1):
        if abs(pos1[i] - pos2[i]) > consts.POS_EQ_THR:
            return False
    # The 2 positions are equal only when all joints are equal.
    return True


# initialize a csv data file with headers
def init_csv_file(file_name: str, headers: list[str]):
    with open(file_name, 'w', newline='') as csv_file:
        writer = csv.writer(csv_file, delimiter=",")
        writer.writerow(headers)


# append multiple lines to a csv data file
def write_csv_file(file_name: str, data: list[list[float]]):
    with open(file_name, 'a', newline='') as csv_file:
        writer = csv.writer(csv_file, delimiter=",")
        writer.writerows(data)


# read the headers and data from a csv file
def read_csv_file(csv_file_name: str):
    with open(csv_file_name, 'r') as csv_file:
        csv_reader = csv.reader(csv_file, delimiter=",")

        # read headers and initialize data arrays
        cdf_headers = next(csv_reader)
        cdf_data = [[] for i in range(len(cdf_headers))]

        # read data
        for row in csv_reader:
            for idx, one_data in enumerate(row):
                cdf_data[idx].append(float(one_data))

    return cdf_headers, cdf_data
```

A.5.- consts.py

This is the Python code with some of the used constants (”consts.py”) that is used by both Slave programs.

```python
import math

# Broker information
BROKER_IP = "192.168.100.173"
BROKER_PORT = 1883
BROKER_USER = "testbed"
BROKER_PASSWD = "1234"

# target arm joints topics
TOPIC_TGT_PREFIX = "joints_val"
TOPIC_TGT1 = TOPIC_TGT_PREFIX + "1"
TOPIC_TGT2 = TOPIC_TGT_PREFIX + "2"

# current arm joints topics
TOPIC_CUR_PREFIX = "current_val"
TOPIC_CUR1 = TOPIC_CUR_PREFIX + "1"
TOPIC_CUR2 = TOPIC_CUR_PREFIX + "2"

# arm ready topics
TOPIC_RDY_PREFIX = "ready"
TOPIC_RDY1 = TOPIC_RDY_PREFIX + "1"
TOPIC_RDY2 = TOPIC_RDY_PREFIX + "2"

# grip topics
TOPIC_GRIP_PREFIX = "Grip"
TOPIC_GRIP1 = TOPIC_GRIP_PREFIX + "1"
TOPIC_GRIP2 = TOPIC_GRIP_PREFIX + "2"

# The QoS level of MQTT messages
QOS_LEVEL = 0

```

```
# RTDE check frequency
RTDE_FREQ = 57  # unit: Hz

# The threshold that we use to check whether 2 joint positions are the same
POS_EQ_THR = 0.000175 * 5
'''
# Our threshold is 0.01/180*pi = 0.000175 (this is our error of position in
    radians of every joint)
# This is because the robot joint usually changes the 0.01 degree, it is not
    stable
# We will do it with the threshold 5 times
'''

# joints positions that compose the path of Arm 1
ALL_POSITIONS1 = [
    [-1.57, -2.007, -2.269, -2.007, -1.57, 1.57],
    [0.5, -2.007, -2.269, -2.007, -1.57, 1.57]
]

# joints positions that compose the path of Arm 2
ALL_POSITIONS2 = [
    [-1.57, -2.007, -2.269, -2.007, -1.57, 1.57],
    [0.5, -2.007, -2.269, -2.007, -1.57, 1.57]
]

# the index of the arm positions at which we need to operate the grippers
POS_IDX_GRIP = 6

# constants about csv data files
DATA_FILE_ARM1 = "data_arm_1.csv"
DATA_FILE_ARM2 = "data_arm_2.csv"
CSV_DATA_HEADERS = [
    "Controller sends target", "Arm receives target", "Arm sends current",
    "Controller receives current"
]
ARM1_NTP_OFFSET_FILE = "arm1_ntp_offset.csv"
ARM2_NTP_OFFSET_FILE = "arm2_ntp_offset.csv"
```

```
CONTROLLER_NTP_OFFSET_FILE = "controller_ntp_offset.csv"
```

A.6.- mir_api_s.py

This is the Python code with some of the used constants (”mir_api_s.py”) that is used by the ”mqtt_Slave_mobile.py” program.

```python
import requests
import json
import math
import time

import urllib3

class MiR():

    def __init__(self):

        self.headers = {}
        self.headers['Content-Type'] = 'application/json'
        self.headers['Accept-Language'] = 'en_US'
        self.headers['Authorization'] = 'Basic
    RGlzdHJpYnV0b3I6NjJmMmYwZjFlZmYxMGQzMTUyYzk1ZjZmMDU5NjU3NmU0ODJiYjhlNDQ4MDY0MzNmNGNmO
    =='
        #
    YWRtaW46OGM2OTc2ZTViNTQxMDQxNWJkZTkwOGJkNGRlZTE1ZGZiMTY3YTljODczZmM0YmI4YTgxZjZmMmFiN
    =='
        self.group_id = 'mirconst-guid-0000-0011-missiongroup'
        self.session_id = '85cd7f3f-f2b7-11ea-ad20-0001299f16e3' #'a2f5b1e6-
    d558-11ea-a95c-0001299f04e5'
        self.local_link = "https://192.168.12.20/api/v2.0.0/"
        self.external_link = "https://192.168.100.51/api/v2.0.0/"
        self.mission_dict = {
            "demo-start": "00e6b5b0-0628-11ef-b2f8-20a7870098ae",
            "demo-placement": "80597d2b-0627-11ef-b2f8-20a7870098ae",
            "demo-finish": "0c4fc706-0628-11ef-b2f8-20a7870098ae",
            "demo-relstart": "e8459301-0720-11ef-bf68-20a7870098ae",
            "demo-relmove": "ced8d6a8-06cd-11ef-bf68-20a7870098ae"
        }
        urllib3.disable_warnings()

    # get the system information
    def get_system_info(self, mir_ip):
        result = requests.get(mir_ip + 'status', headers=self.headers,
verify=False)

        return result.json()

    # get all missions
    def get_all_missions(self, mir_ip):
        result = requests.get(mir_ip + 'missions', headers=self.headers,
verify=False)

        return result.json()

    # get missions
    def get_specific_mission(self, mir_ip, guid):
        result = requests.get(mir_ip + 'missions/' + guid, headers=self.
headers, verify=False)

        return result.json()

    # get actions of a missions
    def get_actions_of_mission(self, mir_ip, guid):
        result = requests.get(mir_ip + 'missions/' + guid + '/actions',
headers=self.headers, verify=False)

        return result.json()

    # get all maps infomation
    def get_maps(self, mir_ip):
        result = requests.get(mir_ip + 'maps', headers=self.headers, verify=
False)

        return result.json()

    def get_specific_maps(self, mir_ip, guid):
        result = requests.get(mir_ip + 'maps/' + guid, headers=self.headers, verify=False)

        return result.json()

    def get_register(self, mir_ip):
        result = requests.get(mir_ip + 'registers', headers=self.headers, verify=False)

        return result.json()


    # get specific map by the map name
    def get_map_positions(self, mir_ip, map_name):
        result = requests.get(mir_ip + 'maps/' + map_name + '/positions', headers=self.headers, verify=False)

        return result.json()

        # get positions details

    def get_all_position(self, mir_ip):
        result = requests.get(mir_ip + 'positions', headers=self.headers, verify=False)

        return result.json()

    # get positions details
    def get_specific_position(self, mir_ip, guid):
        result = requests.get(mir_ip + 'positions/' + guid, headers=self.headers, verify=False)

        return result.json()

    # get a specific guid from the name of a position
```

```python
    def get_position_guid(self, mir_ip, name):
        positions = self.get_all_position(mir_ip)
        for item in positions:
            if item['name'] == name:
                guid = item['guid']
                break

        return guid

    # post a new mission
    def post_mission(self, mir_ip, name):
        parameters = {"name": name, "hidden": False, "group_id": self.
group_id, 'session_id': self.session_id}
        post_mission = requests.post(mir_ip + 'missions', json=parameters,
headers=self.headers, verify=False)
        print(post_mission)

        return post_mission.json()

    # post actions to mission
    def post_action_to_mission(self, mir_ip, mission_id, position_id,
action_type):
        parameters = {'action_type': action_type, 'mission_id': mission_id,
        'parameters': [
        {'id': 'position', 'input_name': None, 'value': position_id},
        {'id': 'cart_entry_position', 'input_name': None, 'value': 'main'},
        {'id': 'main_or_entry_position', 'input_name': None, 'value': 'main'
},
        {'id': 'marker_entry_position', 'input_name': None, 'value': 'entry'
},
        {'id': 'retries', 'input_name': None, 'value': 10},
        {'id': 'distance_threshold', 'input_name': None, 'value': 0.1}],
        'priority': 1}

        result = requests.post(mir_ip + 'missions/' + mission_id + '/actions
', json=parameters, headers=self.headers, verify=False)

```

```
        return result.json()

    # post position
    def post_position(self, mir_ip, name):

        system_info = self.get_system_info(mir_ip)
        position = system_info['position']
        pos_x = position['x']
        pos_y = position['y']
        orientation = position['orientation']
        map_id = system_info['map_id']

        parameters = {"name": name, "pos_x": pos_x, "pos_y": pos_y, "orientation": orientation, "type_id": 0,
                      "map_id": map_id}
        post_position = requests.post(mir_ip + 'positions', json=parameters, headers=self.headers, verify=False)

    def post_to_mission_queue(self, mir_ip, mission_id):
        mission_id = {"mission_id": mission_id}
        post_mission = requests.post(mir_ip + 'mission_queue', json=mission_id, headers=self.headers, verify=False)

        return post_mission

    def get_mission_queue(self, mir_ip):
        result = requests.get(mir_ip + 'mission_queue', headers=self.headers, verify=False)

        return result.json()

    def get_spe_mission_from_queue(self, mir_ip, queue_id):
        result = requests.get(mir_ip + 'mission_queue/' + queue_id, headers=self.headers, verify=False)

        return result.json()

```

```python
    # pause robot executing
    def put_state_to_pause(self, mir_ip):
        parameters = {"state_id": 4}
        requests.put(mir_ip + 'status', json=parameters, headers=self.headers, verify=False)

    # start executing
    def put_state_to_execute(self, mir_ip):
        parameters = {"state_id": 3}
        requests.put(mir_ip + 'status', json=parameters, headers=self.headers, verify=False)

    # start mir
    def put_state_to_start(self, mir_ip):
        parameters = {"state_id": 1}
        requests.put(mir_ip + 'status', json=parameters, headers=self.headers, verify=False)

    # start shutdown
    def put_state_to_shutdown(self, mir_ip):
        parameters = {"state_id": 2}
        requests.put(mir_ip + 'status', json=parameters, headers=self.headers, verify=False)

    # Abort Mission
    def put_state_to_abort(self, mir_ip):
        parameters = {"state_id": 6}
        requests.put(mir_ip + 'status', json=parameters, headers=self.headers, verify=False)

    # get the details of a mission
    def get_mission_guid(self, mir_ip, name):
        missions = self.get_all_missions(mir_ip)
        for item in missions:
            if item['name'] == name:
                mission = self.get_specific_mission(mir_ip, item['guid'])
                break

        return mission['guid']

    # function for deleting missions by name
    def delete_mission(self, mir_ip, name):
        missions = self.get_all_missions()
        result = None
        for item in missions:
            if item['name'] == name:
                guid = item['guid']
                result = requests.delete(mir_ip + 'missions/' + guid,
headers=self.headers, verify=False)
                break

        if result == None:
            result = 'No mission named {0}'.format(name)

        return result

    # function for deleting positions by name
    def delete_position(self, mir_ip, name):
        positions = self.get_all_position(mir_ip)
        result = None
        for item in positions:
            if item['name'] == name:
                guid = item['guid']
                result = requests.delete(mir_ip + 'positions/' + guid,
headers=self.headers, verify=False)
                # break
        if result == None:
            result = 'No position named {0}'.format(name)

        return result

    # get details of a specific mission's actions
    def get_details_mission_actions(self, guid):
        actions = self.get_actions_of_mission(guid)
        len_actions = len(actions)
        i = 1
        for item in actions:
            guid = item['parameters'][0]['value']
            result = self.get_specific_position(guid)
            i = i + 1
            text = 'text'

        return text

    # create a mission with a certain name and return it's guid
    def create_mission(self, name):
        result = self.post_mission(name)
        print(result)
        mission_id = result['guid']

        return mission_id

    # create an action in a specific mission with a default action_type move
    def create_action(self, mission_id, position_name, action_type='move'):
        all_position = self.get_all_position()
        for item in all_position:
            if item['name'] == position_name:
                guid = item['guid']
                break
        # create an action with the specific type
        result = self.post_action_to_mission(mission_id, guid, action_type)

        return result, guid

    # calculate distance in meters between two points
    # def cal_distance(self, origin, dist):

    #     return geodesic(origin, dist).meters

    # get the current position of the robot if accessible otherwise return
None
```

```
    def get_current_position(self, mir_ip):
        sys_info = None
        tries = 0
        while tries < 10:
            try:
                sys_info = self.get_system_info(mir_ip)
                origin = (sys_info['position']['x'], sys_info['position']['y'], sys_info['position']['orientation'])
                return origin

            except KeyError:
                print('Retrying to get information!', 'Time: ', time.time())
                time.sleep(0.1)

        return None

    # get the position defined on the map with the shortest eucledian distance to the robot
    def get_nearest_position(self):
        origin = self.get_current_position()
        all_positions = self.get_all_position()
        best_distance = float("inf")
        cloest_location = ''
        for item in all_positions:
            if item['name'] != 'Config position':
                position = self.get_specific_position(item['guid'])
                dist = (position['pos_x'], position['pos_y'])
                temp_distance = self.cal_distance(origin, dist)
                if temp_distance < best_distance:
                    best_distance = temp_distance
                    cloest_location = item['name']

        return (best_distance, cloest_location)

    # check if the destination has been reached by comparing current position to a specific position
    def check_reach_des(self):
        origin = self.get_current_position()
        all_positions = self.get_all_position()
        best_distance = float("inf")
        cloest_location = ''
        for item in all_positions:
            if item['name'] != 'Config position':
                position = self.get_specific_position(item['guid'])
                temp_distance = math.sqrt(
                    math.pow(
                        origin[0] -
                        position['pos_x'],
                        2) +
                    math.pow(
                        origin[1] -
                        position['pos_y'],
                        2))
                if temp_distance < best_distance:
                    best_distance = temp_distance
                    cloest_location = item['name']

        return best_distance, cloest_location

    # get details of the mission that is currently being executed
    def get_exe_mission(self):
        exe_mission = self.get_mission_queue()
        mission_name = 'None'
        for item in exe_mission:
            if item['state'] == 'Executing':
                mission_gen = self.get_spe_mission_from_queue(str(item['id'
]))
                mission_detail = self.get_specific_mission(mission_gen['
mission_id'])
                mission_name = mission_detail['name']

        return mission_name

    def is_mission_exe(self, mir_ip):
```

```python
        mission_queue = self.get_mission_queue(mir_ip)
        for item in mission_queue:
            if item['state'] == 'Executing':
                return True

        return False

    # Check if there are pending missions in the queue
    def is_mission_pend(self, mir_ip):
        mission_queue = self.get_mission_queue(mir_ip)
        for item in mission_queue:
            if item['state'] == 'Pending':
                return True

        return False

    # Check if a mission is done or not
    def is_mission_done_or_not(self, id):
        exe_mission = self.get_mission_queue()

        for item in exe_mission:
            print(item)
            if (item['id'] == id):
                print(item['state'])
            if (item['id'] == id) and (item['state'] == 'Done'):
                return True

        return False

    def is_latest_mission_done(self,mir_ip):
        exe_mission = self.get_mission_queue(mir_ip)
        if exe_mission[-1]['state'] == 'Done':
            return True

        return False

    # move the mir robot with joystick using continuos velocity messages
    def move_mir(self, mir_ip, state_id, velocity, joystick_web_session_id):
        parameters = {"velocity":velocity}
        requests.put(mir_ip + 'status', json=parameters, headers=self.headers, verify=False)

    # added so the position of the MiR can be changed
    def set_position(self, mir_ip, guid, x, y , orientation):
        parameters = {"pos_x": x, "pos_y": y, "orientation": orientation}
        response = requests.put(mir_ip + "positions/" + guid, headers=self.headers, json=parameters, verify=False)

        return response.json()

    # change the state of the robot manually from (1 to 5)
    def chang_manual(self, mir_ip, state_id):
        parameters = {"state_id": state_id}
        requests.put(mir_ip + 'status', json=parameters, headers=self.headers, verify=False)

    # Send mission to mir
    def set_mission(self, mir_ip, GUID):
        data = {"mission_id": GUID}
        response = requests.post(mir_ip + "mission_queue", headers=self.headers, json=data, verify=False)

        return response.json()

    # Delete the mission queue
    def delete_mission_queue(self, mir_ip):
        response = requests.delete(mir_ip + "mission_queue/", headers=self.headers, verify=False)

    # Delete a specific mission based upon mission_id
    def delete_specific_mission(self, mir_ip, guid):
        response = requests.delete(mir_ip + "mission_queue/" + guid, headers=self.headers, verify=False)

        return response.json()

    def move_to_coordinate(self, mir_ip, x_dst, y_dst,ori):
        move_to_coordinate = {
        'mission_id': '3eb49381-face-11ed-a1eb-000e8e98416b',
        'parameters': [
      {'input_name': 'x', 'value': x_dst},
        {'input_name': 'y', 'value': y_dst},
        {"input_name": "ori", "value":ori}
        ],
        'message': '',
        'priority': 1
        }
        response = requests.post(mir_ip + "mission_queue", headers=self.
headers, json=move_to_coordinate, verify=False)

    def move_to_pos(self, mir_ip, mission_id, pos_guid):
        move_to_pos = {
        'mission_id': mission_id,
        'parameters': [
      {'input_name': 'pos', 'value': pos_guid}
        ],
        'message': '',
        'priority': 1
        }
        response = requests.post(mir_ip + "mission_queue", headers=self.
headers, json=move_to_pos, verify=False)
        # print(response)

    def set_desired_speed(self, mir_ip, speed):
        body = {'value': speed}
        response = requests.put(mir_ip + "settings/2078", headers=self.
headers, json=body, verify=False)

    def set_path_deviation(self, mir_ip, deviation):
        body = {'value': deviation}
        response = requests.put(mir_ip + "settings/2070", headers=self.headers, json=body, verify=False)

    def set_maximum_distance_from_path(self, mir_ip, dist):
        body = {'value': dist}
        response = requests.put(mir_ip + "settings/1769", headers=self.headers, json=body, verify=False)

    def set_waiting_for_obstacle(self, mir_ip, sec):
        body = {'value': sec}
        response = requests.put(mir_ip + "settings/2069", headers=self.headers, json=body, verify=False)

    def post_pos_to_path(self, mir_ip, path_guid, pos_guid, pos_type):
        parameters = {
        "path_guide_guid": path_guid,
        "pos_guid": pos_guid,
        "pos_type": pos_type,
        "priority": 1
         }
        response = requests.post(mir_ip + 'path_guides/' + path_guid + '/positions', json = parameters, headers = self.headers)

    def get_map_guid(self, mir_ip, name):
        result = self.get_maps(mir_ip)
        for item in result:
            if item['name'] == name:
                guid = item['guid']
                break

        return guid

    def get_all_path(self, mir_ip, map_id):
        response = requests.get(mir_ip + 'maps/' + map_id + '/path_guides', headers = self.headers)
        return response.json()

```

```python
    def get_path_guid(self, mir_ip, map_id, name):
        response = self.get_all_path(mir_ip, map_id)
        for item in response:
            if item['name'] == name:
                guid = item['guid']
                break

        return guid

    def post_zone(self, mir_ip, map_name, zone_name, type, cell_points):
        map_id = self.get_map_guid(mir_ip, map_name)

        body  = {
                        "name": zone_name,
                        "type_id": type,
                        "polygon": [
                                    {
                                    "x": cell_points[0][0],
                                    "y": cell_points[0][1]
                                    },
                                    {
                                    "x": cell_points[1][0],
                                    "y": cell_points[1][1]
                                    },
                                    {
                                    "x": cell_points[2][0],
                                    "y": cell_points[2][1]
                                    },
                                    {
                                    "x": cell_points[3][0],
                                    "y": cell_points[3][1]
                                    }
                                ],
                        "map_id": map_id,
                    }
        response = requests.post(mir_ip + 'zones', headers=self.headers,
    json = body, verify=False)

    def get_all_zone(self, mir_ip):
        result = requests.get(mir_ip + 'zones', headers=self.headers, verify
=False)

        return result.json()

    def get_zone_type(self, mir_ip, name):
        zones = self.get_all_zone(mir_ip)
        result = None
        for item in zones:
            if item['name'] == name:
                result = item['type_id']
                # break
        if result == None:
            result = 'No zone named {0}'.format(name)

        return result

    def delete_zone(self, mir_ip, name):
        zones = self.get_all_zone(mir_ip)
        result = None
        for item in zones:
            if item['name'] == name:
                guid = item['guid']
                result = requests.delete(mir_ip + 'zones/' + guid, headers=
self.headers, verify=False)
                # break
        if result == None:
            result = 'No zone named {0}'.format(name)

        return result

    def is_waiting_for_obstacle(self, mir_ip):
        status = self.get_system_info(mir_ip)

        text = status['mission_text']
        if ("Waiting for obstacles" in str(text)):

            return True

        return False




    def clear_errors(self, mir_ip):
        status = self.get_system_info(mir_ip)

        body = {'clear_error': True}

        try:
            response = requests.put(mir_ip + 'status', headers=self.headers,
 json=body, verify=False)
        except Exception as e:
            print("Fatal error")
```