Politecnico di Torino

Collegio di Ingegneria Informatica, del Cinema e Meccatronica

MSc in Mechatronic Engineering

ROS over the Internet: developing a VPN-based ROS network to enable remote communication using a 3D printed robot hand

Supervisors

Prof. Paolo Prinetto

Author

Cesare Pecone

Supervisors at HotBlack Robotics:
Dr. Ludovico Orlando Russo
Dr. Gabriele Ermacora

March 2018

Abstract

Cloud Robotics is a recent paradigm in robotics that leverages on Internet-based technologies to enhance robotics applications. It provides several benefits for the development of robots and robotic applications, such as autonomous mobile robots, cloud medical robots and assistive robots. Cloud robotics requires cloud robotic platforms that can be used by robotic developers to build distributed robotic architectures and to interact with robots using standard Internet based protocols.

Aim of this Master Thesis is to study the main Cloud Robotics concepts and the cloud robotic platform that implements them. Then, a robotic distributed architecture is developed, making usage of the considered cloud platform. Finally, such distributed architecture is validated through the implementation on a service robotics application: the PARLOMA project.

Acknowledgments

I would like to mention all those who helped me in writing this thesis with suggestions, criticisms, and comments: my gratitude goes to them, even though I am responsible for every mistake present in this work.

First, I would like to thank my Professor Prinetto for giving me the opportunity to work on a philanthropic project such as PARLOMA, Ludovico and Gabriele whose support and wise guidance make my ideas become the thesis you can now read. When I grow up I want to be like you, guys!

Some special thanks go to my colleagues and friends who have encouraged me or who have spent part of their time to read and discuss with me the drafts of this work, especially Leo and Serafina.

Thanks to all the wonderful people near and far that I was really lucky to meet and have been a part of my wonderful adventure: to all my roommates met during my stay in the university dormitories, to the so-called "studiamici", "bomberamici", "mensamici" and to all the "dynamites" which I had the pleasure of playing with.

Finally, I would like to thank the people closest to me: my family. My parents, who have always supported me, my brothers and brothers-in-law who have always taken care of me and my fabulous nephews who keep the fire of youth burning in me.
Thanks to my grandparents, who have always believed in me, particularly to my grandfather, Cesare, to whom this work is dedicated.

Contents

List of Tables

List of Figures

Listings

Chapter 1

Introduction

Robotics is a growing area in engineering and innovation fields. In recent times, robotics applications have become increasingly popular and their market is skyrocketing. They were initially used to substitute workers in unsafe situations or to replace them in repetitive and boring tasks. Nevertheless, the huge technological progress have taken them out of simple industrial applications, in fact robots are now becoming part of our everyday life.
This tendency has given rise to **Service Robots**, that are, in agreement with the IFR (International Federation of Robotics), robots that work in part or completely autonomously to provide useful services for the wellbeing of humans, excluding production operations [17]. Several applications are included in Service Robotics: telepresence, housekeeping and cleaning or even education.

Due to the technological advancements, the global service robotics market is in continuous expansion, as shown in Figure 1.1.

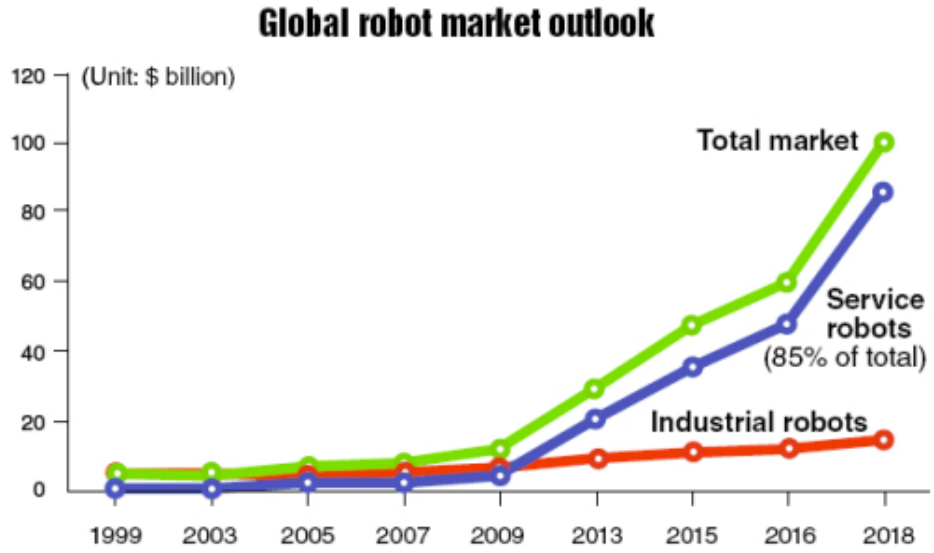


Figure 1.1: Global robots market: service robots vs industrial robots

The service robotics is expected to reach a CAGR (Compaund Annual Growth Rate) of 22% in the years 2014-2020. [1].

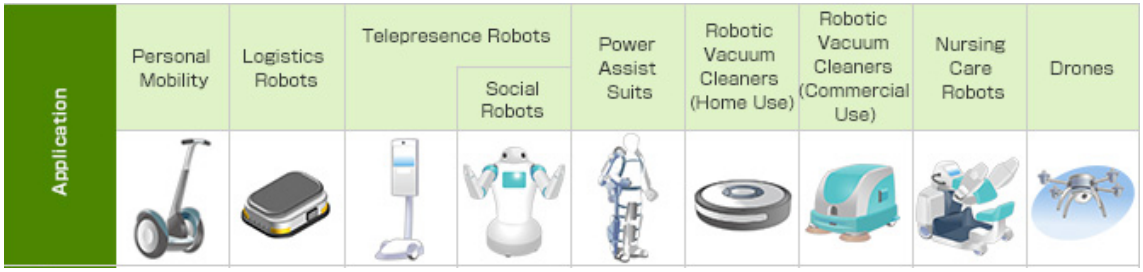


Figure 1.2: Example of Service Robotics applications

Another IT area is growing in recent years that is **Cloud Computing**. In according with the NIST (National Institute of Standards and Technology), cloud computing is a model that enables on-demand network access to a shared pool of configurable computing resources (e.g. servers, networks, storage, services and applications) that can be quickly released and provided with the minimum management effort or interaction with the service provider [16]. This architecture brings numerous advantages in terms of cost reduction, scalability, and huge computing force. It is directly adopted by different applications: e-mail services, social networks, shared document applications, and others. Next, from the fusion of Robotics with Cloud Computing, a new paradigm has been developed: **Cloud Robotics**.

The most comprehensive definition of cloud robotics has been presented by Mario Tremblay, who affirms that: "Cloud Robotics occurs when we connect robots to the Internet and, in doing so, they become more intelligent and capable. The cloud allows them to communicate with other devices and machine, in this way, robots serve their human users more effectively. By collaborating with other devices and people, robots surpass their physical limits and become more capable and useful, since they can delegate portion of their activities to more suitable parts "[13].

So, thanks to cloud robotics, robots can take advantage of the characteristics offered by cloud computing such as cloud storage, distributed computing, access to big data, scalability, coming even to collective learning and crowdsourcing robots (see Figure 1.3).

Many tools can be used to enable Cloud Robotics solutions. In fact, the purpose of this thesis is the following: finding and implementing tools such as ROS, OpenVPN and Docker in order to improve the Parloma project. PARLOMA was created by the collaboration between Politecnico di Milano and Politecnico di Torino, thanks to the Alta Scuola Politecnica (ASP), and developed by L. Russo, G. A. Farulla, and others colleagues, under the supervision of Professors P. Prinetto and B. Bona. This solution will be used to improve the Hotblack Robotics Platform, that is a cloud robotics platform developed by HotBlack Robotics[2], which was founded in 2015 by Dr. Ludovico Russo and Dr. Gabriele Ermacora. In particular, their help played a key role in the development of this Thesis project.

[1] Webcast: Investing in Robotics for 2015 (robotics business review), https://www.roboticsbusinessreview.com/manufacturing/webcast_investing_in_robotics_for_20151/

[2] HotBlack Robotics website, http://www.hotblackrobotics.com/

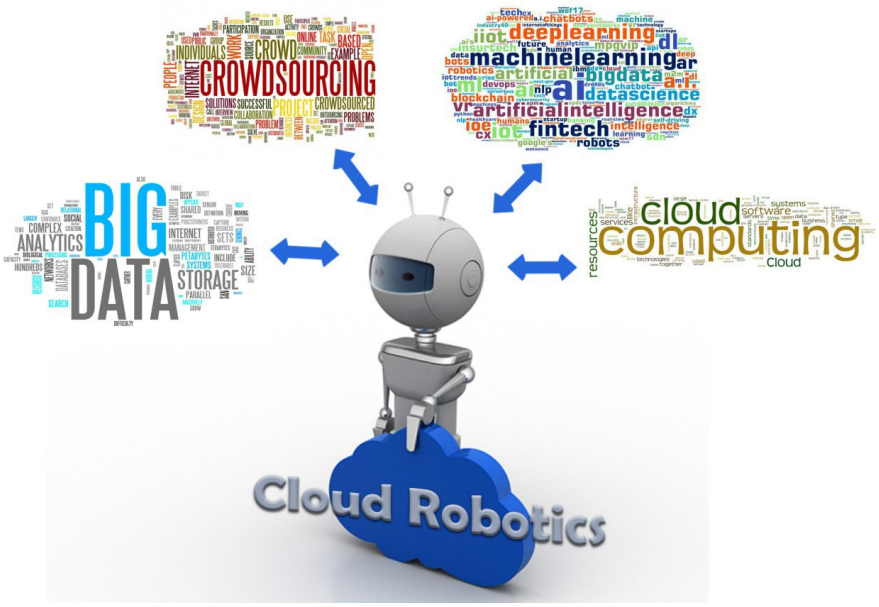


Figure 1.3: Cloud Robotics concepts

1.1 Contextualization

This Master Thesis has been developed under the supervision of Prof. Paolo Prinetto, from October 2017 to March 2018, in the Robotic Research Group (RRG) laboratory of Politecnico di Torino. In addition, since a collaboration with the HotBlack Robotics group was established, also Dott. Ludovico Russo and Dott. Gabriele Ermacora helped in the project development. This Thesis represents the Final Project of the Master Degree in Mechatronic Engineering, and covers different topics studied within this degree, together with the author's personal experience and background. Among the subjects associated with this work, the main ones are: Computer Science, Robotics, Software Design and Development, Operative Systems and Operating-system-level virtualization, Networks.

1.2 Main Objectives and Results

The main goal of this Thesis is the study and development of a network system allowing to use a 3D printed robotic hand through Internet. To reach such objective, different strategies have been considered and discussed during the project. The starting point of this Thesis is the current state of the art of Parloma project in addition with what had been developed in the cloud robotics field by the colleague dott. Pietro Lorenzetti. He allowed the communication between an open-source robot and a cloud robotics platform, creating an ad-hoc VPN network for a ROS Multi-machine

environment.

The Author has implemented a VPN solution, in the ROS environment together with Docker, making the project more portable, flexible and multi-platform.

The strategy is divided in the following steps:

1. A model representing PARLOMA's robot hand has been researched and adapted in a ROS environment. In this way, robotic hand movements can be shown in a 3D visualization system like RVIZ (see subsection 3.1.1).
2. An ad-hoc VPN network has been created between an AWS Server and a host PC. On both of them the ROS framework has been installed, proving the possibity to extend the ROS environment outside of a LAN (Local Area Network).
3. Since the robotic hand should work as a sort of telepresence system, it is needed a sort of packaging, in a way that any user can start a conversation in a user-friendly manner. For this purpose the Docker potentialities have been exploited.
4. Some tests have been done on the network latency, to understand whether the system is reliable for this purpose or not. Then the solution has been tested directly on the hand, and, by means of a Leap Motion device, the acquired data have been forwarded to ROS environment producing the movement of the robotic hand.

The project has led to many useful results both in terms of Parloma project improvements and usability and in terms of enhancements for the cloud robotics platform on which this system will be deployed.

The main Thesis outcomes can be summarized as follows:

- The VPN Multi-Machine ROS network has showed excellent results both in LAN (Local Area Network) and in WAN (Wide Area Network). Surely this solution will be at the center of further developments, aimed at improving its usability, in order to integrate it also on HotBlack Robotics cloud platform.
- The PARLOMA project has been successfully improved and now it is almost ready to be made available to end users.

1.3 Outline

This thesis is organized in seven Chapters as described below.

The first chapter provides a brief presentation to this Master Thesis, introducing a review on the main arguments addressed during the development of the project: Robotics, Cloud Computing, Service Robotics and Cloud Robotics. Furthermore, it provides a description of the work carried out, highlighting the main results and the steps followed to achieve them.

The second chapter contains a general overview of the basic notions needed to well understand the goal of this Thesis: Robotics, Tele-operated Robots, Cloud Computing and Cloud Robotics. Furthermore, some explanations and examples are presented to simplify some concepts.

The third chapter provides main features of the software application used to reach the purpose of the thesis such as: ROS (Robot Operating System), VPN (Virtual Private Network) and OpenVPN and Docker (an operating-system-level virtualization software tool).

The fourth chapter presents the solution adopted. Practically the Author will describe in details how to combine together VPN, ROS and Docker in order to obtain a portable and flexible VPN-based ROS environment. A step-by-step description of the implementation process is provided.

The fifth chapter presents the main features of the state-of-the-art of the PARLOMA project, along with the Authors' contributions to the open issues.

The sixth chapter presents three experiments. The latency of the network system is tested in three different configurations: client/server on a single PC, client/server on a LAN, and client/server on a WAN.

The seventh chapter presents a conclusion to this Thesis work, including achieved results, open issues, and possible future developments.

Chapter 2

Robotics in the Cloud

Nowadays, there is a growing scientific interest in cloud computing. This is a topic that does not only affect engineers, scientists and who work with it, but it is entering into the common conscience of people. However, even if people perceive that cloud computing is an important and growing area, they do not have a clear idea on what exactly it is. As a first definition, "the cloud" is a term that refers to accessing the software applications, IT (Information Technology) and computers through a network connection.

Actually, the cloud is not a brand new notion. It represents the Internet, along with the set of protocols and standards that allow users to use Web services. The term cloud enhances more the idea of complexity and abstraction that it hides. The difference between Cloud Computing and Internet does not concern the involved technologies, which are the same, but in the way services are implemented and designed. Just now, it is not easy to predict which particular routes will be tracked by the cloud, but in the meantime, it is essential to study its potentialities to keep up with the times that are coming.

This chapter provides a detailed overview of the technologies taken into account during the development of this thesis project. These are:

1. Robotics: after a broad overview of robotics and related applications, the focus will be moved on Service Robotics, the area of interest of this Thesis (Section 2.1.1). Finally the Author will provide a theoretical background on Tele-operated Robots (Section 2.1.2).

2. Cloud Computing: the basic concepts at the base and its potentialities are shown in Section 2.2.

3. Cloud Robotics: the implementation of the cloud computing architecture in the robotics field is carefully explained in Section 2.3.

Several documents and books have been consulted to address the theory presented in this chapter; first of all the book Bible of Cloud Computing [24], [17], [18] and [26] for Robotics and Service Robotics, together with some notes of Robotics of the course of Professor Bona. Moreover, for Cloud Robotics, the knowledge comes from: [14], [13], and [12].

2.1 Robotics

Robotics is a division of science and engineering that deals with the design, construction and use of robots: machines can automatically perform a complex series of actions. In addition, it also includes the study of computer systems for robot control, information processing and sensory feedback. Robotics is an interdisciplinary science, as it combines together different engineering fields, along with psychology, biology and philosophy, which address deeper themes associated with robotics. These technologies have been designed and developed to replace human beings in different circumstances such as: manufacturing processes, dangerous environments, and others. Nowadays, the tendency is focusing on service robotics: a new form of robotics linked to everyday life. The latter is at the heart of this chapter and will be covered in detail in Section 2.1.1.

Among all the various applications of robotics, the following are those of main interests:

- **Industrial robots** are manipulators planned to move materials, tools and parts that perform several tasks in production and manufacturing environments.
- **Service robots** are designed to help humans in everyday life. They have a vast range of applications: from floor cleaning to elderly people assistance. Most of the time, they are completely autonomous and planned to operate in a domestic environment. Some examples of Service Robotics are the following:
  - **Agricultural robots** are used for agricultural functions. In the factory settings, they perform different tasks; the main areas are: cloud seeding, weed control, harvesting, planting seeds, soil analysis and environmental monitoring.
  - **Edu-tainments Robots** are used to educate schoolchildren through fun. They can be used to teach some basic concepts in funny way autonomously or led by a teacher.
- **Exploration and scientific robots** also called *rovers*, are designed to explore difficult areas to be reached by humans. In addition, they are also able to collect and analyse samples or send data in real-time to a remote station. Mars rover is a well-known example, it moved itself through the Mars planet.
- **Medical robots** are the ones used in the medical sciences. Surgical robots are the most popular, where robots perform a low-invasive operation while being controlled remotely by an expert.

2.1.1 Service Robotics

The robotics market is constantly expanding. Technological enhancements in software applications and hardware components have made possible many things that were previously unbelievable. These significant improvements have taken robotics to the next step. Whereas before most of the market applications were oriented towards production and manufacturing, now Service Robotics is the new trend. Service robots are defined as: *robots that perform useful tasks for human or equipment excluding industrial automation applications* [17]. In this way, robots will take part in our everyday life: hospitals, hotels, warehouses, restaurants, shops and even in our homes.

Service robots are at a turning point, opening up new contexts for productivity gains more than what industrial ones have done. Advances in engineering areas are allowing robots to be more intelligent, able to work in different environments and capable of complicated manipulations. Also the forces of the ecosystem are reducing the barrier to innovation, involving a wider community of

innovators and making much easier to teach and train robots[1].

As demonstrated by the International Federation of Robotics [18], the statistics clearly show a great increase from 2015 to 2016 (Figure 2.1). Particularly, the number of professional service robots traded in 2015 increased appreciably by 25% to 41.060 units compared to 32.939 sold in 2014.

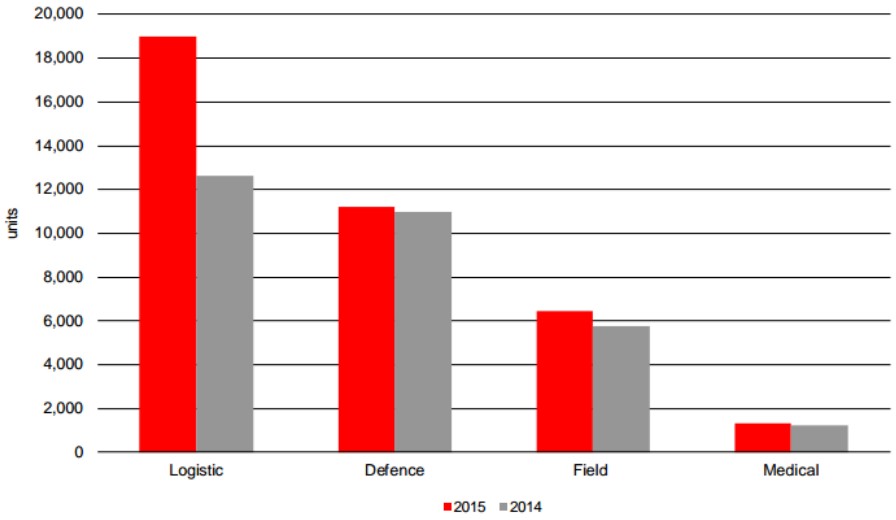


Figure 2.1: Service robots sold units in 2014-15

The projections of the same data in the years 2016-2019 are even more impressive (see Figure 2.2). In fact, rather than the 41,060 units sold in 2015, the total number forecast for 2019 is about 333,200 units with a value of 23.1 billion dollars.

Although the impressive growing trends, some preconceptions still remain about service robotics. The first one is that workers will be replaced by service robots. Actually, this is a wrong belief, since these new robot applications on the market are oriented towards supplementing work in critical tasks/ bottleneck situation or improving health outcomes by making jobs more pleasant. For this reason, the value proposition for service robotics is to support workers to work safer, faster and better.

To understand why robotics is moving toward Service Robotics, an interesting correlation can be determined between new and old paradigms related to Information Technology and Robotics. In table 2.1, the comparison is shown.

[1]Service robots: The next big productivity platform, http://usblogs.pwc.com/emerging-technology/service-robots-the-next-big-productivity-platform/

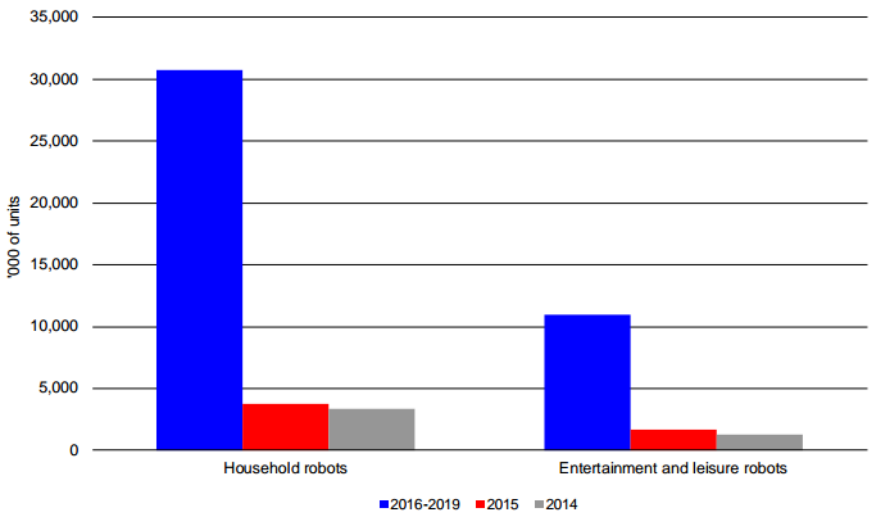


Figure 2.2: Service robots units sales for domestic/personal use, 2016-19 compared to 2014-15

According to [26], several are the challenges that need still be addressed within the service robotics sphere. Below are some of them:

- Making the robots easier to use and program, and therefore involve a larger group of users and innovators;
- Offering robots the skills to understand, perceive, and act in a wide spectrum of dynamic environments;
- Improving robot manipulation ability to handle a wide range of tasks;
- Reducing the size and costs of the robots;
- Allowing robots to collaborate with human, while keeping everyone safe.

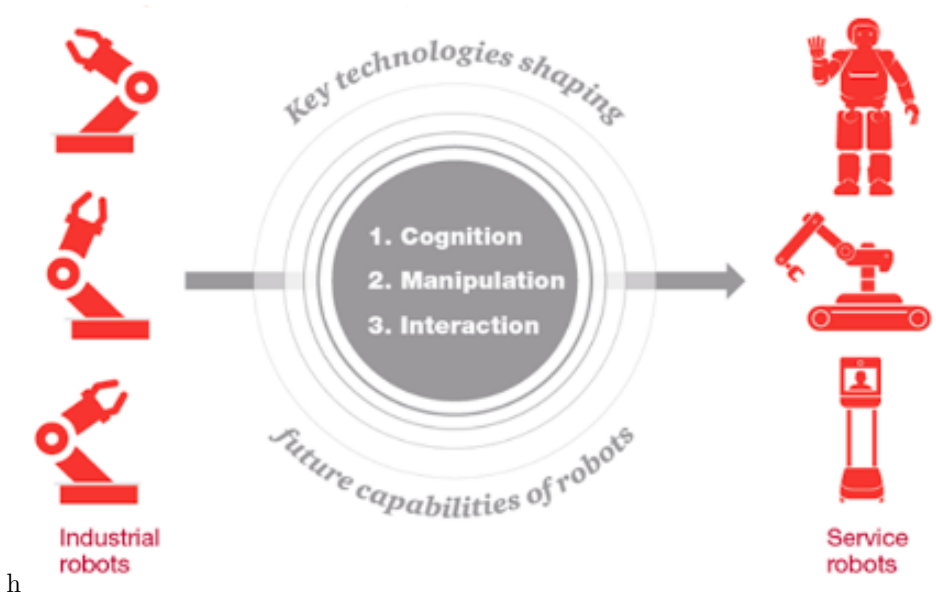
Enterprise IT *old*	**Enterprise IT *new***
Desktop Computers	Many types of end user devices: PC, laptop, tablet, smart-phone
Installed applications	Most applications accessed remotely
Wired in place	Ubiquitous wireless
Specialist team on site	Less need for specialist team

Robotics *old*	**Robotics *new***
In the factory: relatively small number of robot families	More robotic variants with more behaviours
Configured on site	Simpler configuration: potentially via web browser
Fixed positions/tasks	Adaptable devices
Specialist team on site	Most support remote
Wired connectivity	Ubiquitous connectivity: back to supplier/service

Table 2.1: Concepts shift in Robotics and IT in the last years

In order to overcome these challenges, a community effort is needed to improve the following main technologies that are, at the same time, the trend areas that drive the robotics market:

1. **Cognition:** the ability of the robot to understand, perceive, plan and move in the real world. This is the key point to enabling robots to work independently and in complex and different environments.
2. **Manipulation:** proper dexterity and control for handling objects in the environment. Dexterity and manipulation allow robots to execute a wide variety of tasks.
3. **Interaction:** the ability of the robot to collaborate and learn from humans. This kind of interaction will break down the barriers that prevent robots from collaborating with humans.



h

Figure 2.3: Service Robotics Challenges

As better explained in section 2.3, in this scenario, a cloud model called Cloud Robotics plays a crucial role. High computational forces are needed to allow self-learning for robots, advanced algorithms allow robots to learn how to behave in unknown environments, with low or completely absent supervision.
In this way, robots adapt themselves and learn how to face all the possible situations without that programmers have to think about all these. These algorithms require large computational power that cannot be obtained from a single robots or machine. As a result,in order to instruct individual robots, a cloud platform is needed to collect data and to process them. The cohesion of Service Robotics and Cloud Robotics lead to a new paradigm that is **Cloud-based Service Robotics**.

Case Studies

After explaining the main features of Service Robotics, some examples of the real world will be presented, to see how these concepts are actually actualised in real applications.

Fellow Robots[2] produces and developes NVidia Jetson that is an autonomous robot conceived for operating in shopping centres. Here it performs several task, one of these is to monitor the inventory, and if something is missing, inform the employees that will restore it. In addition, it also interacts with customers, guides them to products or simply answers questions.

Figure 2.4: Jetson walking with a customer in a center shop.

Savioke[3] is a robotic enterprise that has created the first delivery robot completely autonomous. It works in different areas, but hospitality is the most profitable. Here it offers amenities, food and an extraordinary experience for guests. Their robot Relay, shown in Figure 2.5, is completely autonomous and able to deliver food and other goods in the client rooms.

[2]Fellow Robots Website: http://www.fellowrobots.com/

[3]Savioke Website: http://www.savioke.com/

Figure 2.5: Relay delivering food in an hotel room.

2.1.2 Tele-operated Robots

Before the robots were networked, the IT world was full of tele-operation systems. They started as devices controlled remotely and then, thanks to the Internet evolution and similar technologies, they have become an integration of human, robots, off-board sensors, cloud and databases all over the world.

The first networked tele-operated robot was created in 1994, by IBM, under the name of "The Mercury Project". It was an industrial robot arm equipped with a digital camera and a little air compressor, it allowed remote users to move on the work space, and to use burst of compressed air to look for buried artefacts in a sand box.
Although most of networked tele-operated robotics devices consist of a single robot and a single human operator; according to Chong et al. [23], there are four types of contexts:

- SOSR: Single Operator Single Robot,
- SOMR: Single Operator Multiple Robots,
- MOSR: Multiple Operators Single Robot,
- MOMR: Multiple Operators Multiple Robots.

This widespread network connectivity considerably increases the potential of networked robots, and enables the use of techniques such as collaborative control and crowd-sourcing as outlined in

Subsection 2.3.1.

There are different degrees of autonomy for robots on the network, the two opposite cases are listed below:

- **Tele-operated**,where through the network, human users send commands and receive feedback. These are just remote controlled robots that do what is required by supervisors.
- **Autonomous**, where through the network, sensors and robots exchange data. In these systems, remote users are not required. The sensing range of the robots is extended by the sensor network.

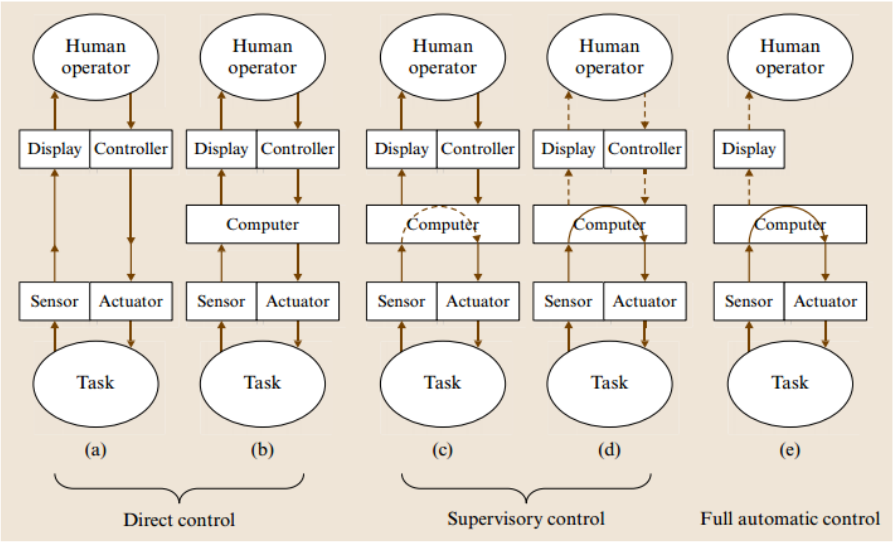


Figure 2.6: Tele-operation control types, from guided to autonomous robots

Figure 2.6 shows the various levels of autonomy of the networked robots.

Networked robotics play a fundamental role in enabling users to interact with remote environment. For instance, in videoconferencing applications, a networked robot can offer much more interactivity, compared to a normal video conferencing system. In fact, a robot could not only represent a remote person, but also transmits multi-modal feedback to the person connected remotely.

In the literature, this technology is called **Telepresence**.

Telepresence

Telepresence is a set of technologies that allow a person to give the impression of being present, or to have an effect, through tele-robotics in a different place from the position of the user. Videoconferencing is the most popular example of telepresence, it consists of streaming video images and audio between two users situated in different locations,an example is in Figure 2.7. A famous videoconferencing software is Skype[4].

Figure 2.7: Videoconferencing example

The advantages of adopting telepresence are many, in working environment for example, travel costs are reduced, the employees work-life is safeguarded and their productivity is optimized.

In addition, users could also perform some tasks in the remote location. Manipulation can be implemented by a robot in the remote location that copies user movements. In this way is possible to control remotely a robot that follows the remote commands while it is streaming audio/video. These functions refer to a new field of robotics: **Mobile Robotic Telepresence (MRP)** or **Telerobotics**. According to [2], Mobile Robotic Telepresence can be defined as *a system that incorporates video conferencing equipment onto mobile robot devices which can be steered and controlled from remote location.* These systems, created to promote social interaction between people, are also becoming prominent in other areas of application such as in office contexts, independent living for the elderly and health care environments. Compared to conventional videoconferencing, MRP provides a deeper social interaction.

[4]Skype Software: https://www.skype.com/en/

2.2 Cloud Computing

The tendency to improve processing and storage techniques, combined with the powerful of the Internet, has made calculation resources less expensive, more available and effective in any place. This trend has led to Cloud Computing.

Cloud term indicates two concepts:

- **Abstraction:** using a layered architecture developers and users are independent from the actual implementation of the system. This feature makes user access, applications and data storage completely intuitive and flexible.
- **Virtualization:** systems are virtualized through resources sharing. This creates the illusion of having more resources that are actually allocated in a real-time model, based on user needs.

The National Institute of Standards and Technology (NIST) provides an accurate definition of this model, that is:

Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model is composed of five essential characteristics, three service models, and four deployment models [16].

From this statement, different key-points come up:

- **On-demand self-service.** A consumer can unilaterally provide calculation capabilities such as server time and network storage, automatically based as needed without requiring human interaction with each service provider.
- **Broad network access.** Capabilities are accessible on the network through standard mechanisms that promote use by client platforms that could be both thin or thick (e.g. mobile phones, tablets, laptops and workstations).
- **Resource pooling.** The computing resources of the provider are grouped to serve more consumers using a multi-tenant model, with different virtual and physical resources dynamically assigned and reassigned depending on consumer demand. The customer has no knowledge over the location of the provided resources so there is a sort of sense of independence in terms of location, even if at higher level of abstraction, it could specify a position (for example, data center, state, or country). Resources could be processing, storage and network bandwidth.
- **Rapid elasticity.** Capabilities can be provided and released elastically, sometimes automatically, to quickly scale outward and inward according to demand. The consumer sees the capabilities as they are unlimited or appropriated at any time in any quantity.
- **Measured service.** Cloud systems monitor and optimize resources usage by leveraging a measurement capability at a level of abstraction appropriate to the type of service (for example, bandwidth, processing, archiving, and active user accounts). The use of resources can be controlled, monitored and reported, guaranteeing transparency both for the supplier and the costumer of the used service.

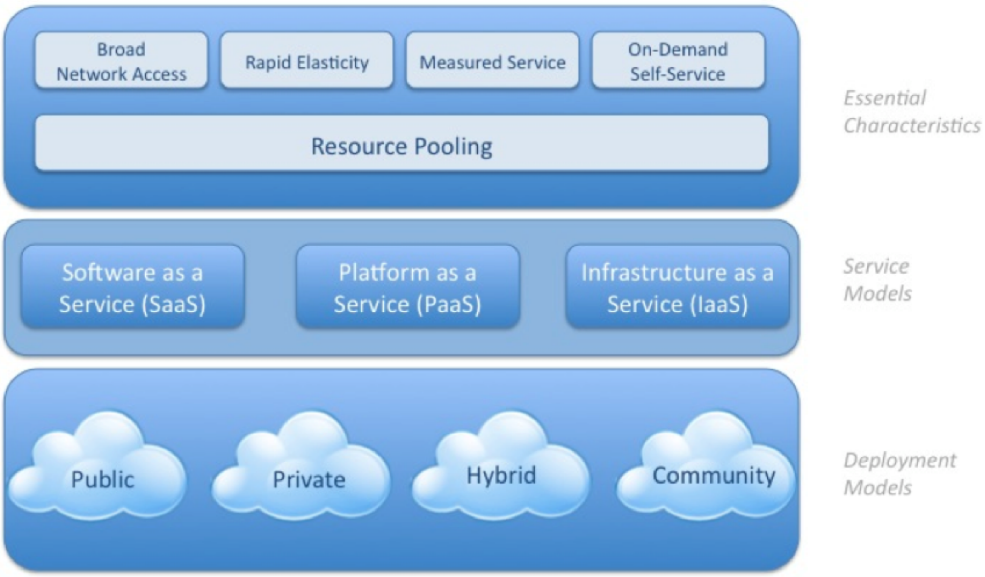


Figure 2.8: NIST definition with the five key points.

2.2.1 Deployment Models

Once the NIST definition is clear, a deep analysis is needed to fully understand how Cloud Computing works. For this purpose, further classifications need to be addressed. The possible distinctions are showed in figure 2.8.

Based on the business scenario and interests of service providers, there are several implementation models that define the nature of the cloud and how it is implemented. Mainly, there are four models:

1. **Public cloud**. A service oriented to the general public that offers different basic characteristics together with many advantages: no initial capital investment and risks transferred to infrastructure providers. However, data controls and security settings are not strict, preventing the use of this model for applications that require a certain degree of security.

2. **Private cloud**. In contrast to the previous one, this model offers the highest level of control in terms of reliability, performance and security. It is designed for private use by a single company.

3. **Hybrid cloud**. A combination of both the private and the public models. Properly, the cloud is working partially in a public environment and partially in private a one. Hybrid Cloud service is very flexible since it is a good compromise between the security levels of private and the benefits of public service. However, they must be carefully designed to accurately divide the private and public parts.

4. **Community Cloud**. This model is generally designed for one or more organisations, which share common purpose in terms of security, policies, mission and so on.

2.2.2 Service models

Before describing the service models, the business model followed by the service providers has to be defined. They offer a service-oriented business model. This means that they propose both hardware and platform resources on demand. This is possible since, as we will see later, the cloud architecture is arranged in such a way that each level can be seen as a service at the higher level and, consequently, as a client of the lower level.

So, there are three main service models adopted by cloud service providers:

- **Infrastructure as a Service (IaaS)**. The whole infrastructure is given to the consumer. It includes virtual storage space, virtual machines and hardware resources that the client can use. On the contrary, the customer implements other deployment aspects such as applications, operating system and so on.
  Real examples of these services are GoGrid [7], Amazon EC2 [4], and Flexiscale [6]. All these companies provide a direct access to hardware resources. The client is then supplied with a VM image as a storage and computer, ready to be filled up with applications and the related operating system.

- **Platform as a service (PaaS)**. This time the customer is only responsible of deploying its applications on top of the infrastructure or to use the tools supported by the platform; while virtual machines, services, operating systems, development frameworks, control structures and transactions are furnished by Paas providers. Real examples of these services are Microsoft Windows Azure [15], Google App Engine [10] and Salesforce [21].

- **Software as a service (SaaS)**. In this model, all the services from the application to the infrastructure takes place on the cloud provider. Mostly the customer accesses it using an interface as a browser and the client's responsibility concerns only with user interaction and data management. When a provider offers software running in the cloud on a pay-as-you-go model, it is referred to as SaaS. Customers can access to the service from their browser, create an account and start using the software without being responsible for installation and maintenance. The SaaS model includes two related but separate concepts: software libraries, which can be imported into other projects as components and standalone software, which can be used on its own. Real examples of this service are Rackspace [11], Salesforce [21] and SAP Business ByDesign [22].

By grouping the three previous service models, the cloud computing SPI (Saas,Paas,Iaas) model is obtained. Many other services exist, such as: IdaaS, Identity as a Service, CmaaS, Compliance as a Service, StaaS, Storage as a Service, and others. However, the SPI model incorporates all of them.

To understand how cloud computing's services are related, it is needed to see the model as a hardware/software pyramid. Observing the Figure 2.9, the set of hardware infrastructures necessary for the cloud is at the bottom, after that, moving upwards, each level inherits the capabilities of the underlying service model.

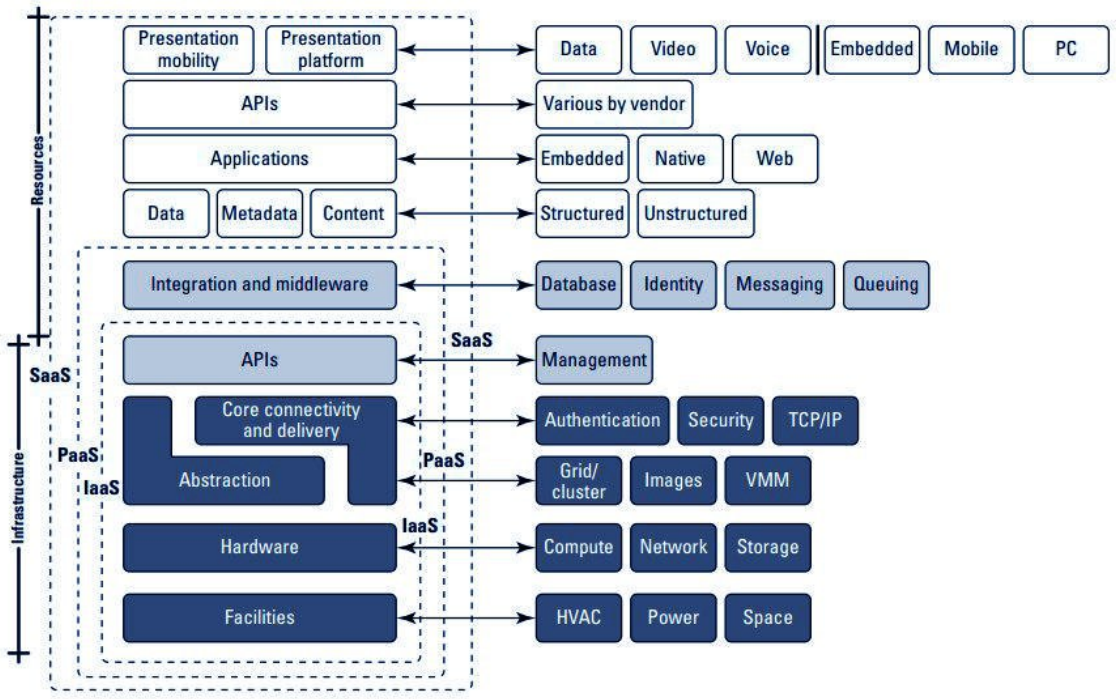


Figure 2.9: The Cloud Computing Stack

2.2.3 Architecture

Before going into details, cloud computing architecture is presented. The structure is divided into four different levels, as showed in Figure 2.10, that are:

1. **Hardware level.** Typically it is implemented in data centers. It is responsible for the whole hardware resource: power supply, router, server and so on.

2. **Infrastructure layer.** Also called Virtualization layer, it partionates the hardware resources using virtualization technologies creating pools of computing units. It is essential as it provides one of the key point of cloud computing: dynamic resource allocation.

3. **Platform layer.** Immediately next to the Infrastructure layer, this layer aims to facilitate the development of applications into virtual machine environments. It consists of application frameworks and operating systems.

4. **Application layer.** It integrates the actual cloud applications. These are featured by the automatic-scaling method, which allows to have better efficiency, availability and performances.

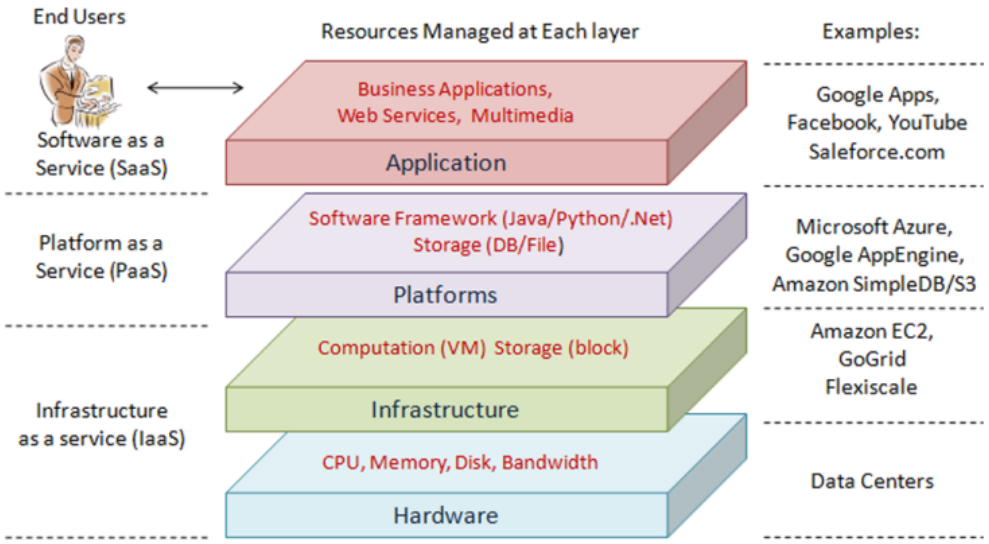


Figure 2.10: Cloud Computing Architecture

The model is highly modular, as a result of this layered architecture. This allows each level to be paired with those above and those below, but also, at the same time, to be able to evolve separately and to be independent. The overall image has been presented, so in order to completely understand the architecture of cloud computing, a more detailed introspection is needed. Composability is the key property of the architecture. Composability is based on two points:

- **Modular.** It is an independent and self-contained unit that is reusable, cooperative and replaceable.
- **Stateless.** A transaction is executed independently of other requests or transactions.

Now three of the layers shown above will be analyzed more precisely: Infrastructure, Platform and Applications.

1. **Infrastructure.** Virtual Servers that run applications are furnished by Iaas providers. These servers share common aspects like memory, CPU cycles, even though they are not physical computers. Figure 2.11 shows the server part of the cloud stack. Generally, together with the Virtual Servers, APIs are also provided, in order to ease the use of their potential from the other layers during the development of applications.
2. **Platform.** The software level used to generate higher levels of service is the Platform. It includes approximately the whole cloud software stack, with the exception of the Application layer that serves as the user interface. Most of the time, platforms are equipped with utilities and tools for collaboration, storing, versioning, testing and so on. Figure 2.12 shows the Platform Layer.

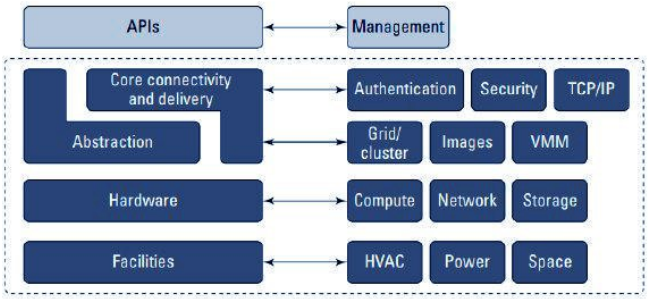


Figure 2.11: Infrastructure part of the Cloud Computing Stack

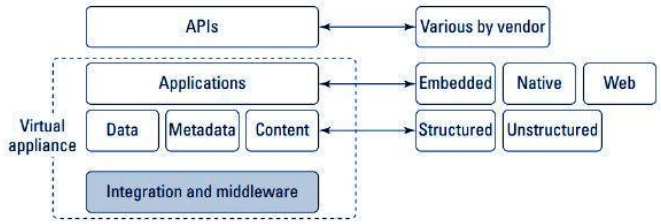


Figure 2.12: Platform part of the Cloud Computing Stack

3. **Application.** The Application Layer is based on all the others, so it is the highest one. It lets users interact by allowing them to use services and to manage their data without caring about how the platform manages services. Usually, this layer goes hand in hand with a User interface that makes easier the use of the application making it more intuitive. Figure 2.13 shows the Application Layer.

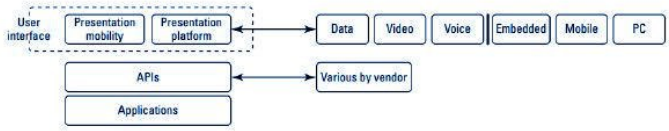


Figure 2.13: Application part of the Cloud Computing Stack

2.3 Cloud Robotics

James Kuffner used for the first time the term "cloud-enabled robotics" during the IEEE RAS International Conference on Humanoid Robots in 2010 [14], highlighting the potential of distributed networks united with robotics.

Following, different definitions have been given, each one concentrating on a several aspect of Cloud Robotics, manifesting the diversification of approaches inside the community. In particular, Ken Goldberg, puts in evidence the possibility of a form of collective intelligence of robots thanks to sharing and learning, resulting from the considerable enhancement of computation capacity, storage, and programming skills: *"Humans as a species are becoming more intelligent because we are able to share information much more quickly and take advantage of innovations faster, even robots have that potential "*[9]. Even Steve Cousins firmly believes that the most significant opportunity provided by Cloud Robotics is the ability to share information between the robots and allowing them to communicate using the cloud, since how is said: *"No robot is an island"* [9].

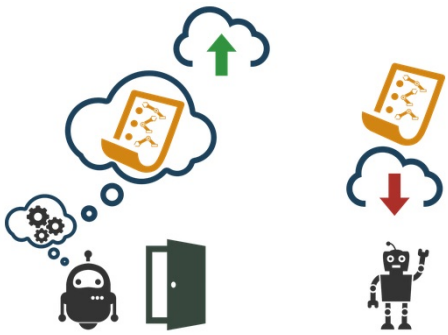


Figure 2.14: Cooperating Robots

One of the most complete definitions was given by Mario Tremblay, who stated: *"Cloud robotics happens when we connect robots to the Internet and then, by doing so, robots become augmented with more capacity and intelligence. The cloud lets them communicate with other machines and serve their human operators better. Connected robots equal augmented robots. By collaborating with other machines and humans, robots transcend their physical limitations and become more useful and capable, since they can delegate parts of their tasks to more suitable parties"* [13].

In technical terms, Cloud Robotics is a robotics area that makes use of cloud characteristics such as Cloud Storage, Cloud Computing, and other Internet features to extend the advantages of shared services and convergent infrastructures to the world of robotics. Powerful computation, communication and storage resources are therefore at the service of the robots, sharing and processing the information collected. This view of robotics makes it possible to create low cost, lightweight and smarter robots, where their computing center is located into the cloud.

Based on the implementation models presented in the Cloud Computing (see subsection 2.2), a new module will be introduced: Robotic as a Service (RaaS), see Figure 2.15. As a general knowledge, RaaS can be explained as a unit of cloud computing that eases the perfect integration

of embedded devices and robots in the cloud computing and Web environment. It follows the Service Oriented Architecture and contains the following features:

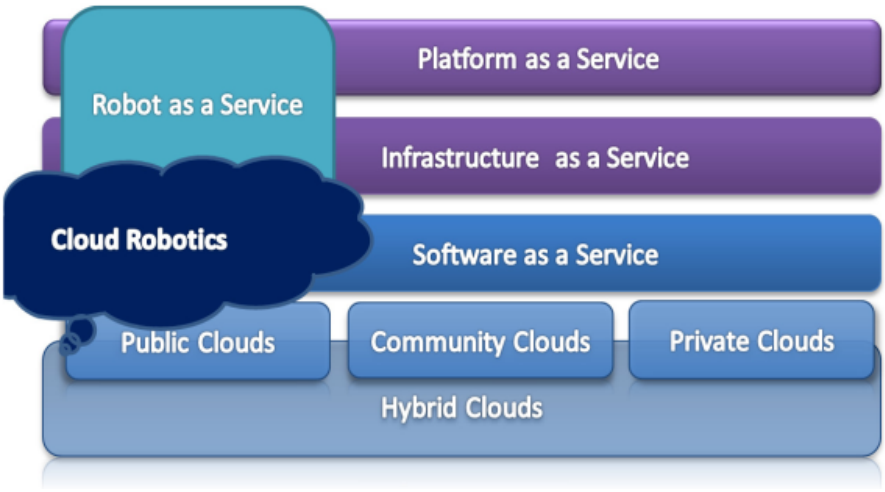


Figure 2.15: Robotic as a Service related to Cloud Computing Stack

- A Raas unit is a service client: according to the available services, new applications can be implemented on the robot.
- A Raas unit is a service provider: it has loaded pre-prepared services that can be modified, removed, or shared with other robots.
- A Raas unit is a service broker: a customer can browse through the several services available and organize them in hierarchy classes.

A basic architecture of Cloud Robotics is composed of two complementary layers:

1. **Machine-to-machine (M2M).** Grouped robots form a cooperative computing entity, which leads to numerous benefits. First of all, as a result of the collaboration, the processing power is greatly increased. Furthermore, information are shared to make collaborative decisions. Finally, the cloud communication range is extended also to the robots that are out of this cloud architecture.
2. **Machine-to-cloud (M2C).** At this level, the cloud provides storage and computing resources that are dynamically allocated depending on real-time demand. Therefore, the heavy calculations and the large amount of data collected by robots can be moved to the cloud.

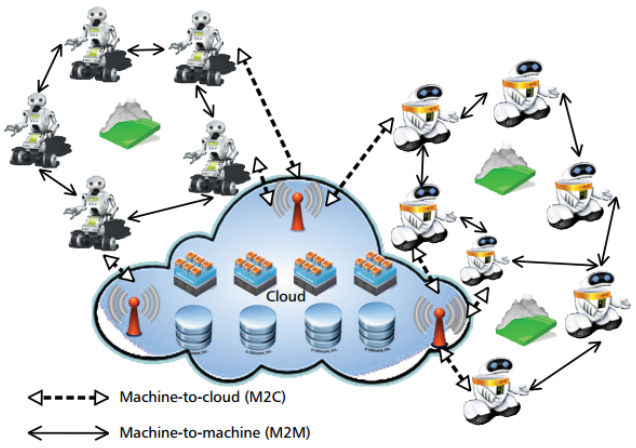


Figure 2.16: M2C and M2M communications that allow robots to be interconnected.

Several are the applications of robotics based on cloud, the most interesting for the purpose of this thesis is the Cloud Network (CN). Cloud Network takes place on the network as an infrastructure and can be subdivided into three groups:

- **Tele-operated Robots.** Robots controlled remotely by a human user. Generally, the goal is telepresence. As explained in 2.1.2, this robotics area has extensive applications in remote exploration, disaster relief, tele-robotic surgery and mining.
- **Multi-robot systems.** Systems in which robots are able to collaborate without any human intervention.
- **Sensor arrays.** They are a form of cloud systems where robots are included in the architecture, and through M2M communication receive information from remote sensors.

2.3.1 Benefits

Once the general concepts of Cloud Robotics have been discussed and some benefits have been briefly cited, it is time to concentrate on the advantages that the cloud offers to robotics.

Big Data. The Cloud Storage allows robotics devices to use Big Data that may not have been kept on internal memory. When we talk about Big Data we refer to a quite wide data sets that could include videos, maps, images, and so on. Big Data together with robotics have played a fundamental role in different examples of applications ; among all, Kehoe in [8] proposed one of the most interesting. It is well known that grasping objects is a usual task faced by many robots, from industrial to service robotics. Actually, it is not easy, for a robot, to implement all the possible procedures that start from the observation of the object to its proper grasping; particularly when it deals with an object never seen before. To this end, Kehoe conceived a system, that, integrating Google Goggles recognition with a sampling-based algorithm is able to recognize objects and suggest grasping path. Practically, the robot takes a picture of the object, sends it to a server; here a trained algorithm compares the photo to a large collection of CAD models. Once the object is recognized, the server returns to the robot, the recommended approach to grasp the

object. Finally, the robot performs the best approach among those proposed and sends a feedback to the server that will be updated. The system is shown in Figure 2.17.

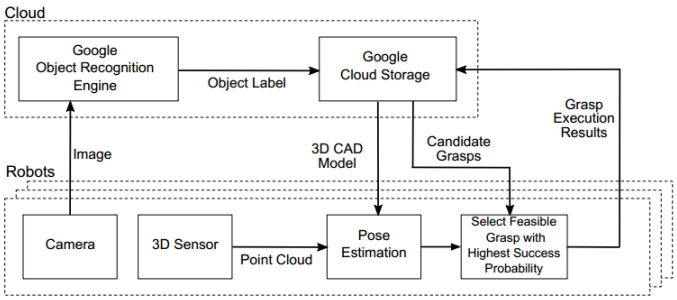


Figure 2.17: Intelligent grasping system architecture

Cloud computing. Having the ability to perform parallel and complicated calculations is crucial in robotics. The power of Cloud Computing has two significant consequences for robots:

- First, it is more important for a robot to be equipped with an Internet connection allowing it to use the external power of Cloud Computing, instead of installing huge processors on board. In this way cost will be cutted-off effectively.
- Secondly, since heavy hardware is moved to the cloud, robots can execute the same functions as before, but being smaller and lighter. This consequence should not be neglected since a lot of applications require small and light robots. Thus even low-cost robots are able to accomplish heavy computational tasks such as functions related to kinematics, motion planning, dynamics and more.

Obviously, this is possible only if the latency of the network is negligible and the quality of the service (QoS) is reliable.

Collective robot learning. Thanks to the Cloud, that gives the ability to share information among the robots and to send feedback between robot and servers, two important benefits can be identified:

- The cloud database exploits not only information coming from a single robot, but from a shared units that collect useful data.
- Once the data are acquired, learning and fusion method can elaborate and enrich the information available.

Waze or Google Maps are examples of Collective Learning which provide vehicle route planner. They collect a series of information retrieved from Maps, Imagery, Satellites, and so on, in this way they have updated information. Later, different users can be seen as a robot community that continuously provide information on traffic jam or best routes. A machine learning algorithm process these data and select the best route in case of traffic congestion and then provides it to drivers. This is a proper example of merging data retrieved by users and existing data.

Crowdsourcing. In some tasks such as facial recognition, image labelling or learning associations between locations and object labels, the human experience and skill are essential. Usually, the cloud offers the possibility to support the robots remotely, in order to improve them for particular functions. In this scenario, the robots can repeat their work continuously and then understand when they are unable to perform certain functions. In these cases, they request assistance and a remote user helps them by instructing them correctly.

2.3.2 Challenges

In addition to the benefits of the Cloud, there are also some problems that must be addressed and overcome.

1. Firstly, as specified above for Cloud Computing, some of the most important problems concerns privacy and security. Examples of privacy issues occur when robots share sensitive information in the cloud about video, images, or other data related to business or private homes. Since these information are also accessible by others users therefore they must be protected in some way. At the same time, security problem includes possible hacking attacks to robots connected to the cloud, causing harms virtual or physical resource or even to people.

   A case of cloud hacking was performed by the Cockrell Engineering School [12] where one professor, Todd Humphreys, along with his students, demonstrate that spoofing a UAV (Unmanned Aerial Vehicle) is technically and practically possible. This proof caught the attention of the Federal Aviation Administration [USA] which decided to strengthen the laws about the UAV, from that moment it is necessary to be in possession of a license to drive a UAV in the US airspace.

2. Another challenge closely related to Cloud both Robotics and Computing, is the QoS and network latency. Although the Internet has grown extremely in recent years, there are several motivations that can cause increase of latency or loss of speed.

   The challenge is to design adaptive algorithms that change according to the QoS of a network, in such a way to lower slowly when the quality of the connection decrease. Alternatively, some customized communication channels may be designed for the sole purpose of connecting robot into the cloud. This problem occurs primarily for real-time applications where hard real-time calculations are required. While, when a quick response time is not required, it does not represent a unsafe limit. For example, in applications as path planning, real-time response is not needed, so network latency doesn't represent a big deal.

3. Finally, a huge problem related to Big Data and Cloud Robotics, is to identify unreliable information and dirty noise. Trusting the wrong information could cause dangerous damage to both data treated and robots, for this reason is essential to be able to get from all available information, only the reliable ones . Usually, the Data Mining algorithms are used to handle noisy information.

2.3.3 Requirements

Following are listed, the minimum system requirements necessary to run a Cloud Robotics architecture.

- First, an Internet connection with a good QoS and as stable as possible is required as a basic requirement. Obviously, the most appropriate connection is the LAN (Local Area Network). However, the latter uses physical cables, it may not be suitable for numerous applications. On the opposite, the WLAN (Wireless Local Area Network) is more adaptable, but sometimes it can be unstable.
- After that the connection is initiated, a calculation unit is needed to accomplish on-board computations and to regulate the cloud communication.
- Next, a suitable software is necessary to form a networking allowing the communication between robots and cloud. In the next chapter, the Robot Operating System (ROS) and other tools will be widely described and also adopted (in 3.1).
- Finally, a platform capable of implementing the required architecture is essential to complete the setting of the environment.

2.3.4 A practical example: RoboEarth

In this section, a Cloud Robotics application in the real world will be illustrated: the RoboEarth project [25].

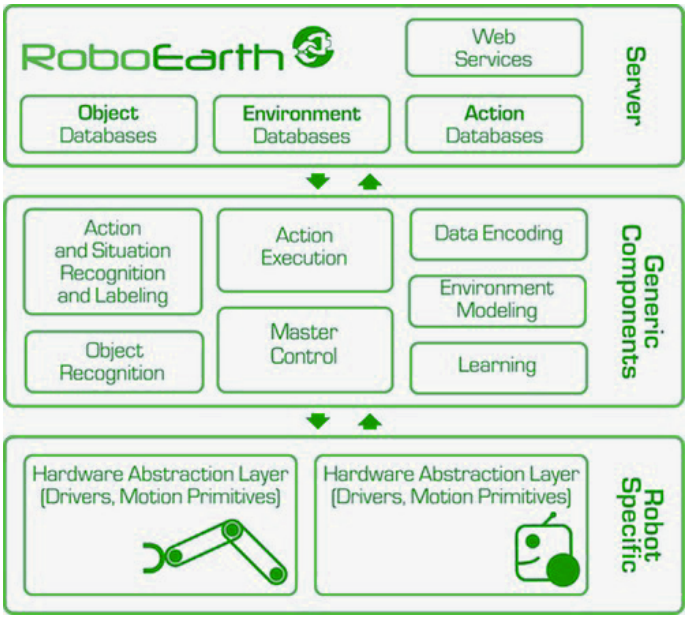


Figure 2.18: RoboEarth Architecture

The RoboEarth project was launched in 2009 and was one of the first application in the Cloud Robotics field. It was born from the cooperation between different universities, that understood the

need to extend computational power of individual robots to a shared processing unit. Furthermore, they have not designed only a shared processing entity, but rather a complete platform (PaaS) in which also data, algorithms, and learning methods are shared. The authors demonstrated two main aspects:

- RoboEarth greatly improves the learning and adaptation of the robot in complicated tasks.
- Using RoboEarth, Robot can perform tasks that have not been explicitly planned in the design phase.

The three-layered architecture, illustrated in the Figure 2.18 represent the core of this project. All the design principles are based on the idea of providing robots the ability to reuse and expand each other's knowledge.

The Server layer represent the bottom of the architecture, that incorporates services and database. A global world model is stored here, along with information on environments, actions and objects. It is possible to access it through common web interfaces. Below it, a Generic Components layer is implemented in the architecture, that makes the entire platform hardware independent. It includes components that are part of the local robot control software and act as an interpreter between robot specific language and RoboEarth directives. At the end, the Robot Specific layer of the robot provides skills abstracting hardware-dependent functionalities from the robot.

Chapter 3

Cloud Robotics: The enabling tools

This chapter will introduce the main features of the software application used to reach the purpose of the thesis. They will be discussed in this order:

1. ROS (Robot Operating System) described in section 3.1. It has been used as foundation of the robotic application developed during this work.

2. VPN (Virtual Private Network) and OpenVPN described in section 3.2. These are the foundations on which the network was created on, allowing communication between multiple ROS machines.

3. Docker described in section 3.3. It is an operating-system-level virtualization software tool that simplifies the task of packing and distributing SW applications, providing portable and flexible systems.

As theoretical background, several sources have been consulted, the main ones are as follows: [5] for ROS, [3] for VPN, [27] together with some notes of Programming for IOT course of Professor Acquaviva for Docker.

3.1 ROS: Robot Operating System

ROS stands for Robot Operating System, even if is not actually an Operating System, it is an open-source framework: a collection of libraries, conventions and tools designed for robotics application developments. It provides the same functionalities of an operating system (for this reason is also called meta-operating system) that contains:

- **Hardware abstraction and device control through driver** means that the code running on ROS is independent from the hardware platform on which it is executed, so for example the velocity command code has always the same form, independently, if the robot system is wheeled, legged or propelled.

- **Multiprocess Communication** means that different programs works in parallel even on different machines, which communicates through an efficient but simple messaging patterns (like publish/subscribe).

- **Package Management** that allow compilation, installation and packages management of ROS-dependent library, data sets, configuration files, or anything else that is usefully organized together .
- **Ecosystem and Community**. ROS is organized in packages and distribution such as is easily to install and use them. ROS provides also many online tutorials and documentation, thanks to the large community behind it, for this reason is becoming day by day more useful and used at the same time. A lot of robot and hardware support ROS, from Nao to ABB industrial robot, from robot arm to rovers.

ROS has to be installed on top of an Operating system. Currently it is supported by Ubuntu or Debian. The environment is written and supports mainly C++ and Python, but ROS has also been designed to be language-neutral so it is possible to use also others programming languages like Javascript, Matlab, Java etc.

However, regardless of programming languages, ROS is composed by three different cores, listed below:

1. **Communication**
   - Each running process is called **node**.
   - Since it is a distributed system, ROS communication is based on a **multi-machines communication**. This means that different ROS processes are distributed on different robots, sensors, actuators or any kind of hardware, which communicate together.
   - The standard communication patterns are Publish/Subscribe, Service/Client and Actions.
2. **Organization**
   - ROS contains codes of **nodes**, libraries, messages, message definitions (how data are packed) and services.
   - These codes is contained inside a **package** that is a collection of nodes, messages and libraries. All related files are encapsulated in the same package. A package can also contain recordings, back files, documentation and so on.
   - In turn, packages could be organized in a **metapackage**, that is a collection of packages thematically similar and somehow interdependent.
3. **Tools**
   - **Command Line Tools**: the fastest interface to visualize and interact with a ROS environment.
   - **RViz**: stands for ROS visualization and allows to see graphically what is going on in the ROS environment, for example a virtualization of a robot that is moving in the real world (see Figure 3.1)
   - **RQT** is a 2D visualizator for data such as graphs representing running nodes and how messages are exchanged (see Figure 3.2)
   - **ROS Bag** is a set of tools for exchanged information exchanged on a ROS application, coming from analysis and debug processes.
   - **Simulators** they provide the necessary interfaces to simulate a robot in a 3D rigid body simulator for robots. Gazebo is the most used simulator.

3.1.1 RViz: ROS Visualization

RVIZ is a 3D visualizer that visualizes what is going on a ROS environment (Fig 3.1).

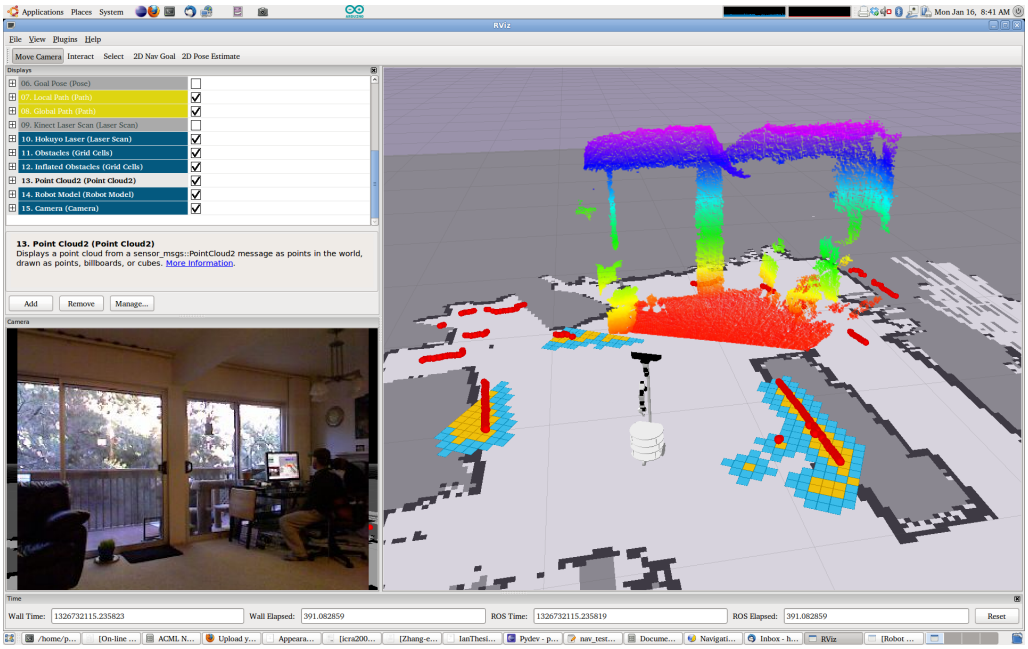


Figure 3.1: RVIZ interface

- The robot is represented on the center of the map.
- Grey part represents the explored floor.
- Black lines represent the walls.
- Red lines are the information obtained by laser scanner.
- Coloured 3D part is what the robot is seeing through the Kinect sensor.

3.1.2 ROS Concepts

As already mentioned, an application developed in ROS is a network of processes able to exchange information in a multi-machines communication network; the **ROS Graph** is a representation of this network (see Figure 3.2) and is composed of the following main parts.

- **Node** is a running process that belongs to a specific package and is identified by an unique name. It is able to exchange messages using topics, actions and services.
  A node can be a driver that controls hardware (like wheels, camera and so on) or a computational node. Generally, information flow starts from a sensor node, passes through a computational node, and ends in a actuator node.

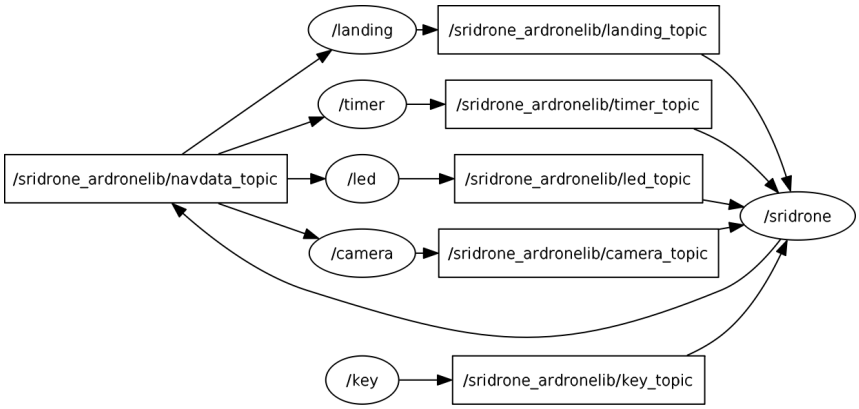


Figure 3.2: RQT Graph of a robotic drone

- **Master** is the main node, it initializes a ROS application and act as the server that enables the communication between other nodes. Every time a new node is launched it will be registered to the ROS master, for this reason the latter is the first node to be executed and, thus, to start a ROS application.
- **Messages** encapsulate data exchanged between topics. Each message has a format described by a **message type**. This format can be simple (bool, int, string, etc.) or a complex structure (vector, matrix, etc.). Each message belongs to a package that could be sensoristic, diagnostic, geometric and so on.
- **Topics** are the main communication channels inside ROS environment, each one is identified by an unique name, each topic is a channel through which nodes exchange information asynchronously (see Figure 3.3).
  - On each topic data can be sent in a specific format (message type).
  - A node that sends data to a topic is called publisher.
  - A node that receives data from a topic is called subscriber.

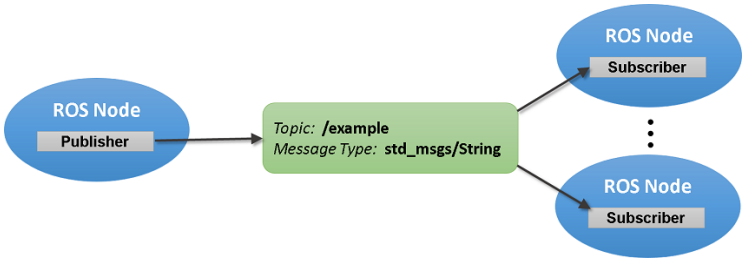


Figure 3.3: Publish/Subscribe communication pattern

- **Service** is another communication channel in ROS, it works like topics but synchronously. This means that the one that sends a message (Client) waits a response from the one that receives received the message (Server). It is a blocked system.
- **Action** is a cross between a topic and a service, since a blocking system does not exist. For this fact, while the "client" is waiting for a response from the server a feedback is received. It is designed to work on complex or long tasks.
- **Parameter Server** Different nodes could share the same parameter called parameter server. Tt is currently part of the master node.

3.2 Virtual Private Network (VPN)

VPN is very useful in a scenario where, for example, a Data Center or headquarters corporate networks need to be connected to remote branches as remote data center, backup storage center, or even to single remote users, for example somebody working by home or by travelling that wants be connected as if he/she are in the office.

Until some years ago, these problems were resolved using dedicated connections or leased lines, generally really expensive. But in the last years, since the internet is ubiquitous and connecting to it is easy and inexpensive, it was thought to use internet to send and exchange data with the corporate network.

It is obvious that due to people's security needs and above all the need to send encrypted data over a network, VPN technology has been developed. But in addition to the role of creating a "private sphere of computer communications", VPN technology has many other advantages:

- High level of security. When a VPN is used, data are kept encrypted and secure.
- Remote control. In an enterprise environment, data center and information can be accessed remotely, from home or anywhere else. So a VPN greatly increases the company's productivity.
- Shared files. It is a good solution for users that want to share data for a long period.
- Better performance. Efficiency and bandwidth of the network increase, generally, where VPN solution are implemented.
- Reduced costs. When a shared resource is used, the costs are shared as well. So it is cheaper since the maintenance cost is very low.
- Provide access to corporate network in a selective and flexible way.
  - Selective. Because only those who have the right profile can connect to the corporate network, moreover, for some external users it is possible to provided limited services.
  - Flexible. Since the solution is based on being interconnected on a shared infrastructure it doesn't matter where the users are.

A VPN is mainly used to extend a private network, which is a network that assigns private IP addresses, through a public network. This allows users to share data on a shared network, as if their machines were connected to a private one. In fact, within the VPN system, IP addresses are assigned to the devices, as if they were private. When it was designed, its sole purpose was to create a network tunnel allowing remote access to machines. However, then the VPNs have been adopted principally to mask IP addresses, for security reasons.

In short, it works as follows. Commonly, when a machine requests an Internet connection, it first connects to the ISP (Internet Service Provider), this provider then forwards it to the requested Web site. In this way, the ISP could keep trace of the Internet traffic that passes through their ISP servers. On the contrary, when a VPN solution is adopted, machines connect to servers by way of the adopted VPN provider guaranteeing an encrypted connection. Thus, IP addresses cannot be tracked by the ISP and data are protected.

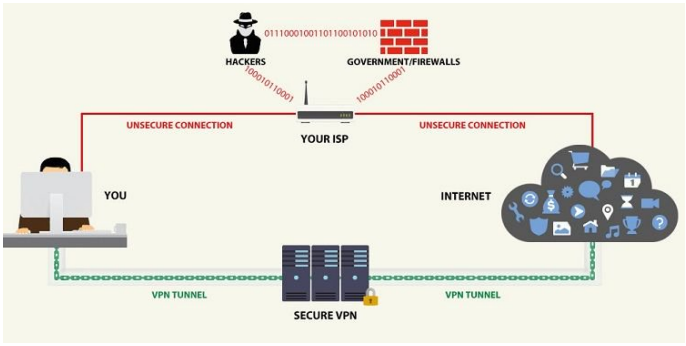


Figure 3.4: VPN vs ISP

The VPN solution has different consequences:

- Data are encrypted so the ISP cannot track the websites visited by the users, because the VPN server routes all the Internet activity. Basically, the ISP sees only the VPN server.
- The VPN server IP address hides the real IP addresses of the machines connected to Internet. This means that the visited websites cannot track the real IP address of the connected machine.
- Data are encrypted only for the ISP provider but not for the VPN server. In fact, all the sensitive data can be accessed from the VPN provider.
- The main disadvantage of using VPN solutions is the network latency. It is principally caused by two aspects: the additional distance that data need to cover for reaching the VPN server, and encryption/decryption of data that require time for processing. These problems have been experienced in this Thesis work.

3.2.1 OpenVPN

OpenVPN is a famous open-source software that provides VPN solutions for building remote access structures and site-to-site or point-to-point secure connections in bridged or routed configurations. VPN uses a custom protocol of security based on Secure Sockets Layer or Transport Layer Security (SSL/TLS) protocols that offer different authentication procedures, such as: username/password system, certificates, certificates or pre-shared secret [1].

In order to achieve the goal of this Thesis, the OpenVPN application has been used to create a client-server configuration (see subsection 4.2), in which the authentication certificates for the clients that use certification authority (CA) and signature are released by the server.

There are several reasons why OpenVPN has been adopted. The main features[2], that differentiate it from other VPN applications are the following:

[1] OpenVPN description, https://openvpn.net/index.php/component/content/article/55-about-openvpn.html

[2] OpenVPN features, https://openvpn.net/index.php/open-source/335-why-openvpn.html

- OpenVPN is an open-source software and it is supported by a powerful community of developers.
- OpenVPN offers high portability, it is compatible with FreeBSD, Linux, OpenBSD, NetBSD, Solaris, Windows and Mac OS X.
- OpenVPN allows the user to significantly customize network attributes. Every setting from protocol to the security countermeasures used can be adjusted according to the needs and requirements of the application.

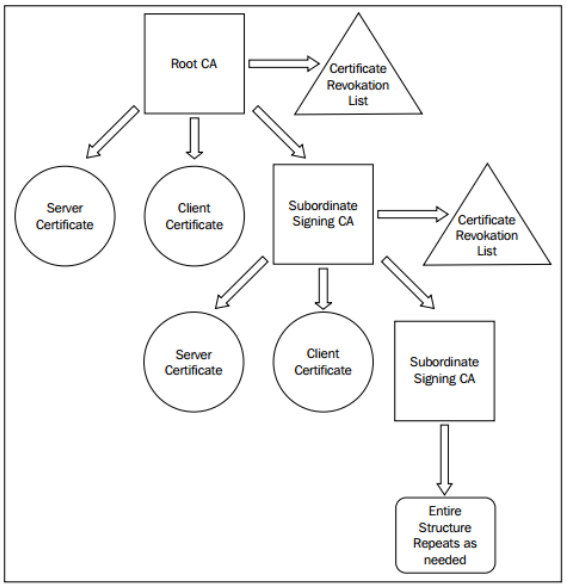


Figure 3.5: PKI (Public Key Infrastructure)

Before going into details, some useful notions have been addressed in order to allow a good understanding of how the server/client solution has been implemented. In particular, the following are of primary interest:

1. The difference between UDP (User Datagram Protocol) and TCP (Transmission Control Protocol) is fundamental. They are both based on the IP protocol and are used to send data over the net. This indicates that whether UDP or TCP is implemented, the packets are sent towards an IP address. Actually, there are some important differences to be taken into account in choosing one or the other. TCP is an end-to-end protocol, it manages error control, packet ordering, flow control and rate control based on packet round-trip time, guaranteeing the arrival of each package. However, redundant retransmission into a congested network could introduce unwanted time delays in a tele-robotics networked system. On the contrary, UDP does not implement a retransmission mechanism. It is a broadcast-capable protocol in which users must manage rate control and error control. UDP packets are sends at the sender's default rate but the speed is changed according to the network congestion. UDP is really efficient since it has less overhead than TCP, but often firewalls block it just because of the absence of a rate control mechanism.

2. The second important notion is the use of the CA (Certificate Authentication) method that increases highly the security of the connection. OpenVPN exploits the *X.509* certificates for the encryption of VPN traffic and for clients authentication. Once that the Certificate Authentication is reliable, it is possible to build a public key infrastructure (PKI) as shown in Figure 3.5. When many users are present on the network, there is a high potential of stolen and lost keys. Therefore, with a properly configured PKI, it is relatively easy to revoke a lost certificate or that of a departing employee. However, for a single point-to-point link, it does not make sense to implement the complexity of PKI just to protect a tunnel; rather it is preferable to use pre-shared keys in this situation.

3. Finally, after configuring the PKI and before starting a trusted connection, it is important to understand the steps that are accomplished by the client and the server. Figure 3.6 shows these steps through a detailed diagram of the operations performed before the connection.

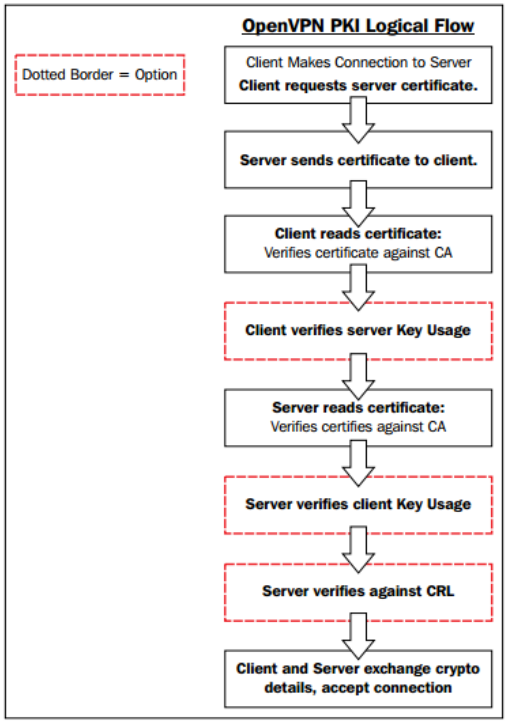


Figure 3.6: Steps performed before starting the connection

3.3 Docker and containers

In the previous sections we discussed about cloud platforms that offer microservices, what are their advantages and why are so used from companies. In this section a practical way to implement them will be explained. The solution is to use containers and in particular the Docker containers.

3.3.1 Containers vs VMs

In computer science Virtual Machines (VM) is a very common software tool mainly used to emulate Operating Systems. Generally on a hardware infrastructure an Operating System is hosted and, when a VM software is used, a component, called hypervisor, divides your hardware and enables multiple Operating systems running on the same infrastructure. The role of the hypervisor is to simulate the fact that multiple machines are running. In this way, different guest OSs can run simultaneously. For example on top of a Windows OS it is possible to run a Linux system and so on. Of course it is possible to develop your own applications in the different guest OS. Each of these ones has its own binaries and libraries and the application will be deployed on these libraries (See Figure 3.7). VirtualBox or VMWare are the software most commonly used.

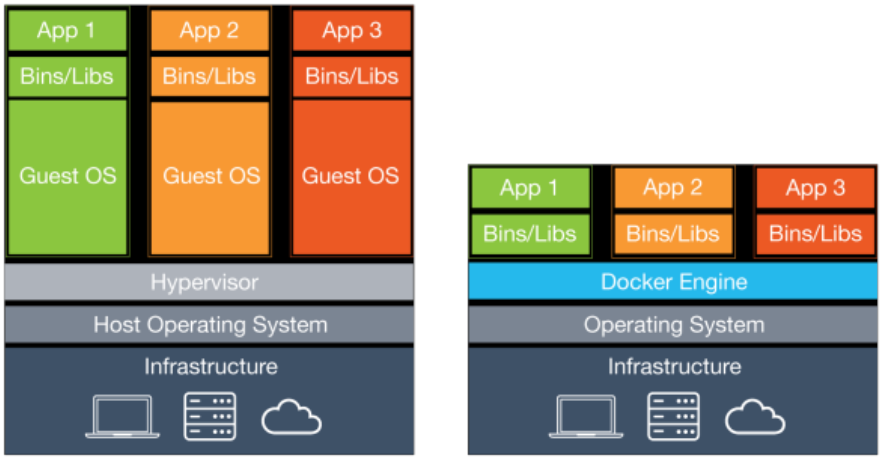


Figure 3.7: Docker vs VM

On the other side, a container works differently, as shown in the figure 3.7: the guest OS is completely skipped and there is a direct link between the Operating system of the infrastructure and the Container Engine. The Container Engine (also called deamon) is the software implementing the containers, and binaries and libraries are directly installed on top of it, without having a whole guest OS that takes a lot of memory, storage and processing from hardware resources. In fact VM wastes a lot of resources to run a complete operating system, because in this type of virtualization, when containers are used, a full image of an OS is uploaded, but this is instead not necessary. So, it is possible to develop applications totally independent from the real operating system and from the hardware infrastructure. Furthermore it is possible to move these applications, without doing any modification, into another system which must support just the container engine. In this way you can easily run an application of a machine onto another with different hardware, without re-installing anything because the container engine already does the job for you. The

main advantage of containers is that they are much lighter than the systems using hypervisors; of course they have limitations consisting in not having available a full OS with related features and advantages. However in terms of developing applications in a completely independent and portable way and, thus for cloud robotics developing, this is exactly what we need.

3.3.2 Docker

Originally the containers idea was introduced in the Linux world where they were known as the LXC. The purpose of these ones was to be able to run multiple isolated systems inside a single machine.
Docker is a container system making use of LXC container. The main difference between Docker and original LXC is that the first is much more dynamic. It is defined *application-centric* because it is optimized for the deployment of applications as opposed to machines. In fact, after that a docker container is enabled, it runs an application, when it finishes the docker engine will be switched off and everything will be removed.

Docker engine guarantees high flexibility: machines can powered up and switched off faster, since the size of these images is very light (more or less the same of the application running on top of it) and so, also the resources used are fewer. Of course there are limitations, in particular, since containers are designed to run just a single application, if n processes have to be executed, also n containers have to run. In fact, only a single application and nothing else can run in background a container.

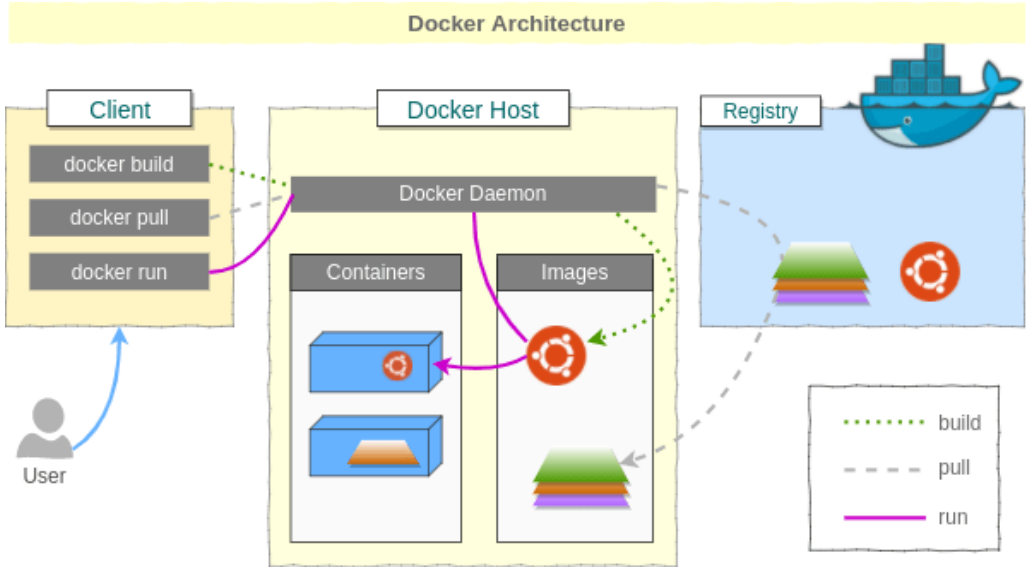


Figure 3.8: Docker Architecture

Docker Architecture

Docker uses a client-server architecture. The Docker client talks to the Docker daemon, which does the heavy lifting of building, running, and distributing your Docker containers. The Docker client and daemon can run on the same system, or you can connect a Docker client to a remote Docker daemon. The Docker client and daemon communicate using a REST API, over UNIX sockets or a network interface.

As shown in the figure 3.8, the Docker Daemon takes a Dockerfile from a register, builds it making an image, and then can run the image. The image concept is similar to the one of the VM, while in the case of VM, the image refers to a full OS (Windows, Ubuntu, etc.), in this case the image represents a set of single applications along with an execution environment. This environment is made of binaries and additional executable (commands that come with libraries).

So, the image is the application plus all the programs and binaries required to run it. It does not contain memory manager, file system manager, visualization tools, setting environments, scheduler etc.

When an image is running, it means that image has become a container, so a container actually is a "live" version of a image. Thus, while image is something static on the hard drive, the container is actually the image that is in execution.

Chapter 4

Multi-Machine VPN-based ROS Network

This chapter addresses the main part of the Thesis project: the creation of a Multi-Machine VPN-based ROS Network, which allows to connect a Cloud Platform to I/O modules hosted into different LAN networks, using a ROS environment through the Internet. This work will greatly improve the PARLOMA project, as explained in Chapter 5 In fact, until now, ROS nodes could communicate just on local ROS networks. Obviously this limitation strongly affects the potential of the Cloud Robotics system making it almost useless, since it is designed to operate remotely. Thanks to the Author's contribution, these limitations have been overcome, allowing the expansion of the ROS architecture over the Internet.

This concept is a key point not only for this project but also for the Cloud Robotics paradigm and it is a fundamental requirement to exploit the full potential of the cloud. As a result of the Author contribution, similar projects could be realized also by means of the HotBlack Robotics cloud platform.

In particular, starting from this work, many advantages are obtained in Cloud Robotics Area:

1. **Tele-operated robotics** can be implemented. So a robotic system can be remotely controlled. In the next Chapter, a direct application will be tested on the PARLOMA robotic hand.
2. **The distributed computing** can be perfectly adopted. Once there are no local area network limitations, heavy computation can be moved on cloud platforms using directly a VPN-DOCKER solution. What is needed is just a stable Internet connection and a *Docker Daemon tool*. For example, considering the heavy computer vision algorithms, they could be performed on the platform or remote server, shifting all computations out from the local I/O hardware devices.
3. In future developments, the **collective robot learning** may be used. After that the architecture is established, many robots can connect to it from all over the globe, sharing data and feedbacks. In this way, the cloud platform database exploits not only the information from a single robot, but from multiple robotic devices that acquire useful data. They can also exchange information and tasks, between them, but this is possible only if they work in the same environment and this thesis work aims precisely at this.

This Thesis work makes it possible to support all these applications through a Cloud Platform. In the next subsection, the architecture of the developed solution will be explained.

4.1 The overall architecture

A proper multi-machine architecture has to be set up to extend the ROS network outside of a local network. The ROS environment provides different tools to initiate a system that uses multiple machines. However, as described in the official ROS guide[1], in order to properly start the network, all the involved machines must satisfy these two requirements:

1. A full bi-directional connectivity is needed between all the machine pairs.

2. Each machine must be identified by a unique name traceable by all other machines.

Typically, the IP addresses are represented as unique names. This means that the involved robots need to be connected to the same LAN to identify each other. Therefore, a loophole is needed to set up a network that can be recognized by ROS as local, even though actually it is not. In this way, the functionalities of ROS can still be exploited, together with the potential of a distributed network.

VPN. There are different alternatives to bypass these limitations, in particular, VPN is the one chosen for the development of this thesis. As explained in section 3.2 a VPN allows to extend a private network, which is a network that assigns private IP addresses, through a public network. This allows users to share data on a shared network, as if their machines were connected to a private one. In fact, within the VPN system, IP addresses are assigned to the devices, as if they were private. In this way, all the machines involved in the ROS environment can be incorporated within the same VPN, assigning each of them a different IP, regardless of their actual LAN. Once each device is represented by a unique IP address, it can be used to refer to the different machines within the ROS environment, simulating a single network. This solution has been adopted, because it is easy to implement and provides a good degree of flexibility. Moreover, through VPN technology, the architecture is also safer, since it is channeled through the Internet. To initiate the VPN architecture, the OpenVPN[2] software application (described also in subsection 3.2.1) has been used. In section 4.2 , the configuration of the VPN solution is described in details.

ROS. After that the VPN has been properly built, the next step is to set up a ROS architecture operating within this network. In order to provide the highest degree of flexibility and versatility, the ROS Master has been executed into the VPN Server. In particular, this servers is initiated inside docker container, that run in a Amazon Web Services (AWS)[3] platform. This AWS platforms are used both as cloud platform for PARLOMA and HotBlack Robotics.

[1] ROS multi-machine network setup, http://wiki.ros.org/ROS/NetworkSetup

[2] OpenVPN features, https://openvpn.net/index.php/open-source/335-why-openvpn.html

[3] Amazon Web Services (AWS), https://aws.amazon.com/it/

DOCKER. As explained in the introduction, a multi-machine ROS architecture through VPN was already developed for a telepresence system in HotBlack Robotics by the collegue Lorenzetti. However this system had two huge limitations:

- The system could work only on Linux systems.
- Only a unique instance of the architecture could be managed. This means that just a single clients/server network can be implemented.

In order to solve these problems, the Author has decided to implement a Docker solution, providing:

- A multi-platform system, since Docker can be installed easily on different machines (PC, Servers, Raspberry and so on)
- Multiple instances of the docker container, permitting to manage multiple networks.
- A mechanism that automates the set up of the VPN tunnel and of the ROS environment.

Right after each network is initiated, three Dockerfile will be built, consequentially two Docker containers will be executed on the client side and one for the server side. The two client containers will know the server address and will be able to connect with it.

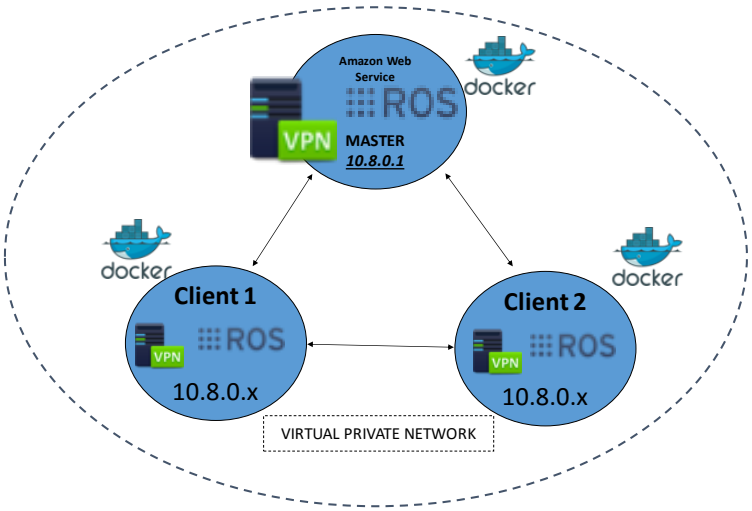


Figure 4.1: Network solution

In the next sections all the tools configurations will be explained in order to get a Multi-Machine VPN-based ROS Network: starting from the OpenVPN tool in Section4.2 then DOCKER tool in Section 4.3 finally ROS framework in Section 4.4.

4.2 OpenVPN Configuration

OpenVPN is a useful tool that allows different configurations. Each model can be customized in terms of transmission protocol, security and more. For the development of this part, the Mastering OpenVPN [3] has been consulted. Firstly, when setting up a VPN a proper deployment model has to be chosen. In this project, since the required configuration must be implemented between a server and multiple clients, the model client/server with TUN (namely network TUNnel) devices has been selected.

After choosing the model, few steps are necessary to build a functioning environment; these procedures will be listed below briefly. Then the client configuration files and the server configuration file adopted will be dealt with in details.

The following steps are:

1. Set up the CA (Certificate Authority) directory and configure the CA variables;
2. Create the Certification Authority (ca.crt, ca.key, key.txt);
3. Creates Server Certificate, Encryption Files and Server Key (dh2048.pem, server.crt, server.key);
4. Generate a Client Certificate and a Key Pair, for each client (client_*.crt, client_*.key);
5. Configure the OpenVPN service by copying all previous files to the appropriate directories: one for the Server and one for each Client (see Figure 4.2);
6. Creates the Server configuration file (server.conf);
7. Creates the Client configuration files (client.conf)(one for each client);

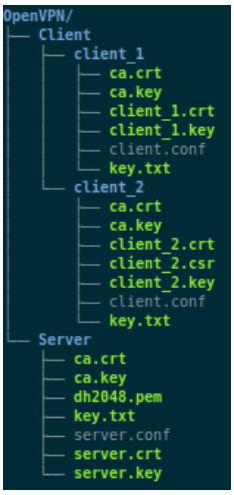


Figure 4.2: OpenVPN directory

8. Start the service.

After these steps, different files will be obtained and will be organized as in Figure 4.2. Since all the files different from the .conf are generated automatically and do not contain any particular parameter, only the files client.conf and server.conf will be explained in details.

Server configuration file.

```
1  port 1194
2  proto tcp
3  dev tun
4  ca ca.crt
5  cert server.crt
6  key server.key
7  dh dh2048.pem
8  server 10.8.0.0 255.255.255.0
9  ifconfig-pool-persist ipp.txt
10 keepalive 10 120
11 tls-auth key.txt 0
12 cipher AES-256-CBC
13 persist-key
14 persist-tun
15 status openvpn-status.log
16 verb 3
```

Listing 4.1: Server configuration file (server.conf)

The configuration file for the server side is shown in Listing 5.7.
Basically it describes all the options that must be taken into consideration when the service is started.
First of all, line 1 defines the port on which OpenVPN should listen to.
In lines 2-3 the *proto tcp* and *dev tun* options are used in place of their counterparts, namely *proto udp* and *dev tap*. As explained in subsection 3.2.1, the TCP protocol is more robust so for this kind of data streaming, in which it is better to avoid loss of information, it is generally preferred.
Other important options are those of lines 4-7, which indicate the name of the authentication files needed to settle a secure TLS (Transport Layer Security) connection.
Finally, a useful information is the one of line 8 which indicates the IP addresses assigned by the VPN, that will be of the kind 10.8.0.x.
The other parameters are almost standard and will not be dealt with in details as they do not have a crucial influence on the functioning of the network.

Client configuration file.

In this architecture server/multi-clients, many clients could be connected to the server. In this case, as illustrated in Figure 4.1 two clients will be involved. Both their configuration files are tailor-made for the same VPN server. So, since they are similar at all, only one of them will be illustrated in details in Listing 4.2.
The first line states that the device will be used as a client inside the client/server architecture.
Next, in lines 2-3, as before, the device typology and the protocol are defined.
The line 4 is also very important, since it indicates the remote address of the server and the related ports that will be used to establish the connection. An actual IP address does not appear since

the automated system, provided by Docker function, substitutes this string with the VPN Server address (see Section 4.4).
As for the server, lines 9-11 show the corresponding paths of the authentication files and keys. In particular for the client_1 the suffix *_1* is used.

```
1  client
2  dev tun
3  proto tcp
4  remote serveripaddress 1194
5  resolv-retry infinite
6  nobind
7  persist-key
8  persist-tun
9  ca ca.crt
10 cert client_1.crt  # or client_2.crt
11 key client_1.key   # or client_2.key
12 remote-cert-tls server
13 tls-auth key.txt 1
14 cipher AES-256-CBC
15 verb 3
```

Listing 4.2: Client 1 configuration file (client.conf)

Service initiation.

After the three configuration files are correctly hosted in the corresponding directories, the Docker containers will initiate the communication and create the VPN tunnel (the procedure is described in next Section). Then, if no errors are present, the connection is built. Note that, during the development of this phase, some problems may encounter from the configuration of the AWS server. In fact, it could present some default firewall settings that block the connection with the VPN tunnel. In particular, a firewall rule, which explicitly allows communication on port 1194, is needed.

4.3 Docker Configuration

In section 3.3, what is a docker container and how does it work is explained. In this subsection, the basic Docker commands, which compose a Dockerfile along with the commands allowing to start a container, are explained.
All the information here reported has been obtained from the official Docker documentation[4].

4.3.1 Dockerfile commands

Docker can create images automatically by interpreting the instructions listed in a Dockerfile.
A Dockerfile is a document that includes the commands a user could run into the command line in order to construct an image. Running the "docker build" command, an automated build, which runs multiple command-line instructions sequentially, is executed.
The Docker daemon executes the Dockerfile's instructions one-by-one, adding the result of each instruction to a new image if necessary, before issuing the new image ID. So each instruction runs independently and causes a new image to be created.
Where possible, Docker reuses the intermediate images to significantly speed up the docker build process.

The main commands that have been used in this project development are the following:

- **FROM**
  Any Dockerfile must start with a *FROM* instruction that specifies the *Base image* from which the new image will be built. The *Base image* is generally pulled from the *Docker Public Repositories*, also called *HUB Docker* [5].

- **RUN**
  The *RUN* instruction executes each command in a new layer above the current image and adds the latest changes to it. The resulting image will be used as base for the next command in the Dockerfile.
  Layering *RUN* statements and commit generation are consistent with the key concepts of Docker, where containers can be built from any point in an image's history and commits are convenient, just like source code control.

- **CMD**
  Only a single CMD instruction could be executed in a Dockerfile. If more CMD are listed, only the last one will be executed. This command represents the default execution of a container. Practically, the CMD instruction sets the first command to execute when the image is running. If arguments are added after the *docker run* command, then the CMD command will be overwritten by the new parameter.
  It is easy to confuse RUN with CMD. RUN actually executes a command and commits the result on the image; CMD does not do anything at image's build time, but specifies the intended command for starting the container.

[4]Dockerfile reference, https://docs.docker.com/engine/reference/builder/

[5]Docker HUB, https://hub.docker.com/

Docker Compose is a Docker tool that runs multi-container applications through the configuration defined in a YAML file. It is very useful even to run a single container because it allows to use a simple command, which extrapolates the configuration information (*port mapping, volumes, tag*) by default from a file called docker-compose.yml and runs the container with all its options.

Now that the basic knowledge about the main commands used in a Dockerfile has been illustrated, the Author will focus on how to use these commands.

4.3.2 Server Dockerfile

The Dockerfile for the server container is shown in Listing 4.3. Basically it contains all the commands needed to start an instance that incorporates an OpenVPN server and a ROS environment.

```
1  FROM osrf/ros:indigo-desktop-full
2
3  RUN \
4   svn checkout https://github.com/creos92/network.git/trunk/Server &&\
5   apt-get update &&\
6   apt-get install build-essential checkinstall -y &&\
7   apt-get install libssl-dev liblzo2-dev libpam0g-dev wget -y &&\
8   wget https://swupdate.openvpn.org/community/releases/openvpn-2.4.4.tar.gz &&\
9   sudo tar zxvf openvpn-2.4.4.tar.gz &&\
10  cd openvpn-2.4.4 && ./configure && make && checkinstall -y
11
12 CMD cd Server && chmod +rx start_s.sh && sync && ./start_s.sh
```

Listing 4.3: Server Dockerfile

- The line 1 is the command that specifies the *Base image* from which the new image will be built, in this case the *Base image* is the *osrf/ros:indigo-desktop-full*. This means that all the ROS commands can be used in the container. It is provided by the official Docker repository and it is maintained by the OSRF (Open Source Robotics Foundation). This *osfr/ros repository* provides different pre-build images that embed several ROS framework releases.
- The command, in line 4, provides the download of the configuration and certification files for the OpenVPN server, described in the previous Subsection 4.2.
- The lines from 5 to 10 are the commands needed to download and install the software packages necessary to run the OpenVPN software tool.
- The line 12, initiates the bash file start_s.sh (see Listings 4.7), that automates the process of creating a VPN server and implementing of ROS.

Then, through the *docker build . -t server* command, the Dockerfile will be compiled resulting in a new image. In order to start a container from this obtained image, the command *docker-compose up* will be executed. The container will be initiated starting from the option configuration listed in the *docker-compose.yml* (illustrated in Listing 4.4)

```
1 server_instance:
2   image: server
3   stdin_open: true
4   tty: true
5   privileged: true
6   ports:
7     - 1194:1194
```

Listing 4.4: Server docker-compose.yml file.

- Line 1 is the command that specifies the tag that will be assumed by the Docker Container.
- Line 2 contains the image that will be used to start the container.
- Lines 3-4-5 indicate the commands useful when it is needed to interact with the started container.
- Lines 6-7 represent the commands that publish the container's ports to the host and viceversa. In this way, it is possible to use the services hosted by that port. In this case, the port corresponds to 1194.

4.3.3 Client Dockerfile

The Dockerfile for the client 1 is shown in Listing 4.5, instead the one for the client 2 is not listed, since it is almost similar. You will notice that many commands correspond with the one of the Server Dockerfile 4.3.

The main differences are explained below:

- Line 4 provides the download of the configuration and certification files for the client 1 side, described in the previous subsection 4.2.
- Line 12 initiates the bash file start_c_1.sh (see Listing 4.8), which automates the process of creating a VPN client and of implementing ROS.

```
1  FROM osrf/ros:indigo-desktop-full
2
3  RUN \
4   svn checkout https://github.com/creos92/network.git/trunk/Client &&\
5   apt-get update &&\
6   apt-get install build-essential checkinstall -y &&\
7   apt-get install libssl-dev liblzo2-dev libpam0g-dev wget -y &&\
8   wget https://swupdate.openvpn.org/community/releases/openvpn-2.4.4.tar.gz &&\
9   sudo tar zxvf openvpn-2.4.4.tar.gz &&\
10  cd openvpn-2.4.4 && ./configure && make && checkinstall -y
11
12 CMD cd Client && chmod +rx start_c_1.sh && sync && ./start_c_1.sh
```

Listing 4.5: Client 1 Dockerfile, the Dockerfile for the Client 2 is almost similar: just replace start_c_1.sh with start_c_2.sh

Through the *docker build . -t client1* command, the Dockerfile will be compiled resulting in a new image. In order to start a container from this new image, the command *docker-compose run -e IP=ServerAddressIP client* will be executed. The IP is the environment parameter that will be obtained from the Cloud Platform and contains the *IP Address* of the Server instance to connect. This IP address will be substituted to the "ipserveraddress" in the *client.conf* file (see Listing 4.2). The container will be initiated starting from the option configuration listed in the *docker-compose.yml* (Listing 4.6).

```
1 client_1:
2   image: client1
3   stdin_open: true
4   tty: true
5   privileged: true
```

Listing 4.6: Client docker-compose.yml file.

Almost all the lines correspond with the docker-compose.yml file of the server, only the name of the container and of the image are different.

4.4 ROS Configuration

After that the clients and server containers are run and the OpenVPN and the Robotic Operating System (ROS) software are installed on it, it is time to exploit their potential.

As anticipated in the introduction of this Chapter, the VPN solution allows the usage of ROS across different networks, instead the Docker solution is needed to automate this configuration.

In particular, the main bash files will be listed, 4.7 ad 4.8 in order to understand which commands are used to start the network architecture. These bash files could be started directly using a CMD command, as seen in the clients/server Dockerfile 4.3 and 4.5.

4.4.1 Server initiation Bash File

```
1  #!/bin/bash
2  chmod +rx /Server/certificati/server/start_server.sh &&
3  cd /Server/certificati/server && sync && ./start_server.sh &
4  sleep 1s
5  roscore &
6  sleep 2s
7  export ROS_IP=10.8.0.1
8  export ROS_MASTER_URI=http://10.8.0.1:11311
9  ifconfig
10 rosnode list
```

Listing 4.7: start_s.sh file

- Line 1 defines which shell should be run with using a shebang. Specifying #!/bin/bash means that the script should always be run with bash. /bin/sh is an executable representing the system shell.
- Command at line 2 is the system call which changes the access permissions to file system (directories and files). In this way it gives the permission to bash file (start_server.sh) to be executed.
- Command at line 3 is one of the most important since it starts the OpenVPN server, through the start_server.sh script, that basically executes the command *openvpn –config server.conf*.
- Line 5 starts a collection of nodes that are pre-requisites of a ROS-based environment. The roscore command corresponds with the initiation of the ROS MASTER. This allows the communication between different ROS nodes.
- Lines 7-8. Once OpenVPN is running, each machine will be able to connect with the others through the VPN IP address. When a ROS node advertises a topic, it provides a hostname (port combination (a URI) that other nodes will contact when they want to subscribe to that topic). The Master can be initiated and its IP indicated as ROS_MASTER_URI can be exported together with the ROS_IP of the machine, in order to set up the multi-machine framework. After the Master initialisation, the other machines can be configured to enter in the same ROS network. To this purpose, we have to export their ROS_IP as well, followed by the ROS_MASTER_URI that is the same for all the client machines and refers to the one of the Master.

- Command at line 9 displays the IP address and the network interfaces. This command allows to observe if the VPN solution is actually activated.
- Command at line 10 permits to see which nodes are running inside the ROS network.

When the Server Docker Container will be run, the output, in Figure 4.3 will be obtained. As planned, the VPN server IP address is *10.8.0.1* and the */rosout* node is running.

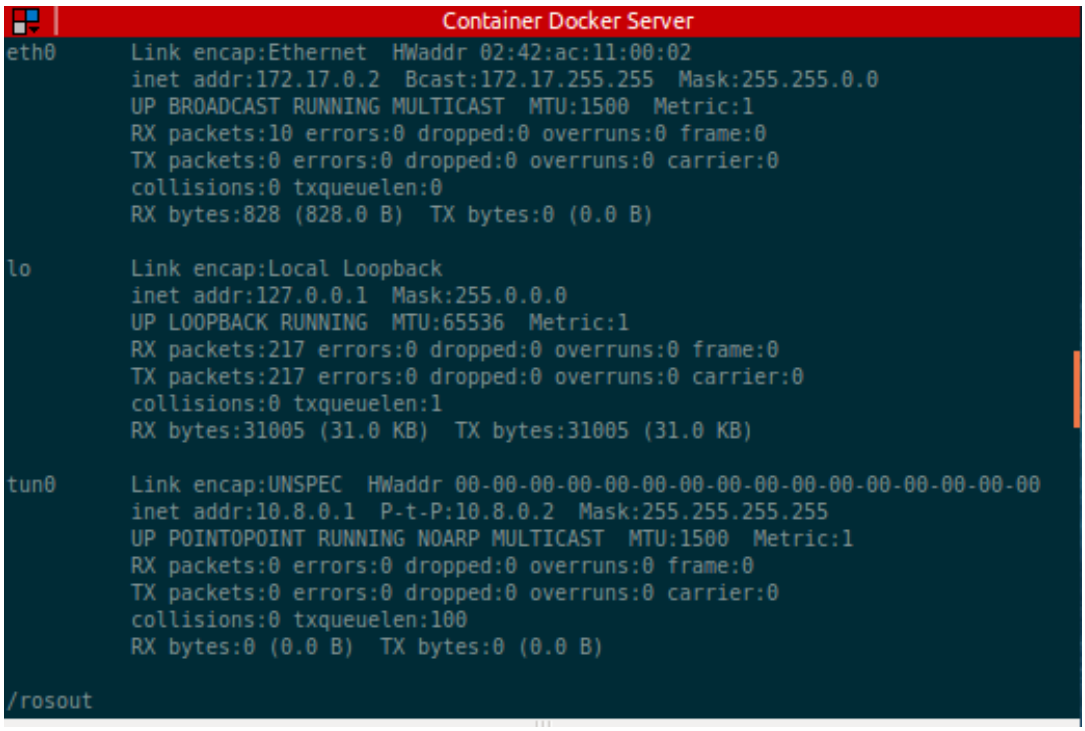


Figure 4.3: Server Docker container Output

4.4.2 Client initiation Bash File

```
1  #!/bin/bash
2  sed -i "s/indirizzoipserver/"$IP"/g" ../Client/certificati/client_1/client.conf
3  chmod +rx /Client/certificati/client_1/start_client.sh &&
4  cd /Client/certificati/client_1 && sync && ./start_client.sh &
5  sleep 3s
6  export ROS_IP=$(/sbin/ifconfig | grep "inet addr:10.8"
7                  |awk -F: '{print $2}' | awk '{print $1}')
8  export ROS_MASTER_URI=http://10.8.0.1:11311
9  ifconfig
10 rosnode list
```

Listing 4.8: start_c_1.sh file.

The start_c_1.sh is very similar to the start_server.sh file, for this reason only the different statements will be illustrated:

- Command at line 2 allows to substitute "ipserveraddress" in the *client.conf* file (see Listing 4.2) with the IP variable obtained when the docker is run through the command *docker-compose run -e IP=ServerAddressIP client_1* .
- Line 4 starts the OpenVPN client, through the start_client_1.sh script, that basically executes the command *openvpn –config client.conf*.
- Command at line 6 extracts the IP address that starts with 10.8.xxx.xxx, resulting by the command *ifconfig* executed inside the container. It uses this IP to uniquely identify the own machine on the ROS environment.

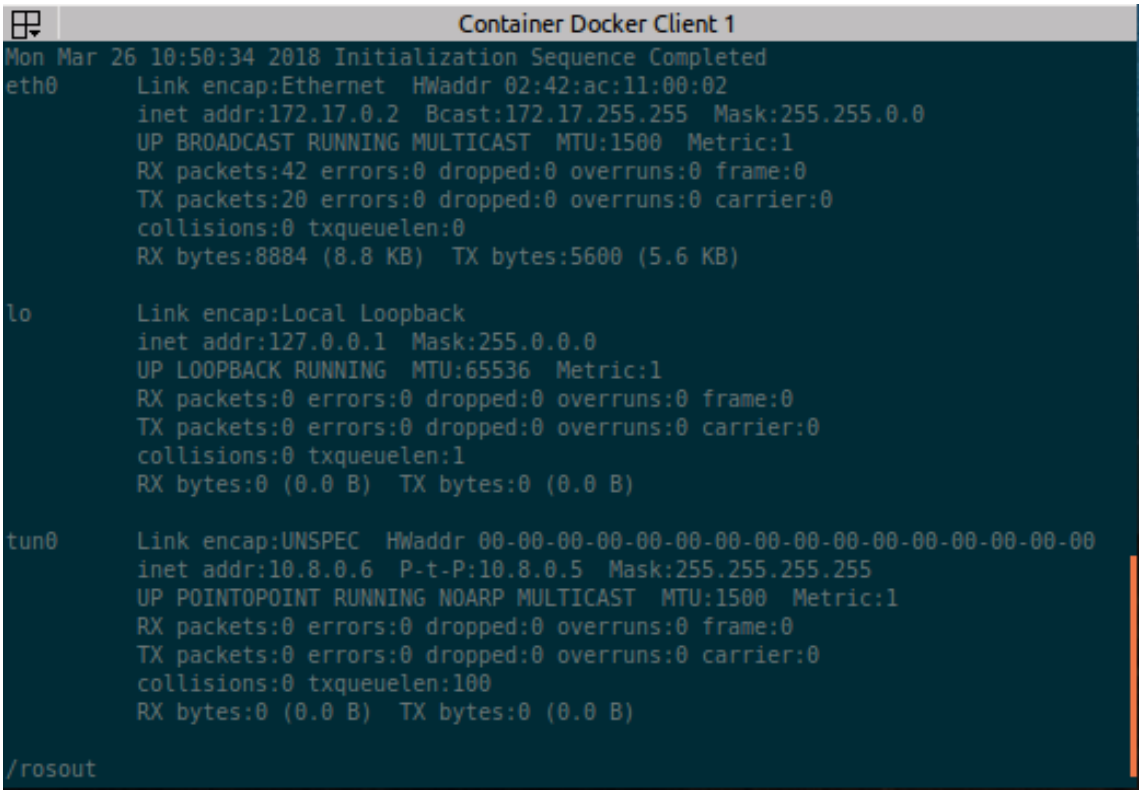


Figure 4.4: Client 1 Docker Container Output

When the Client 1 Docker Container will be run, the output in Figure 4.4 will be illustrated. As expected, the VPN client IP address is *10.8.0.6* and the */rosout* node is reached also by this container.

Client 2 initiation Bash File

As far as the client 2 side is concerned, the start_c_2.sh file is almost similar. The few different statements in lines 3-4-5 are similar: just substituting client_1 with client_2.

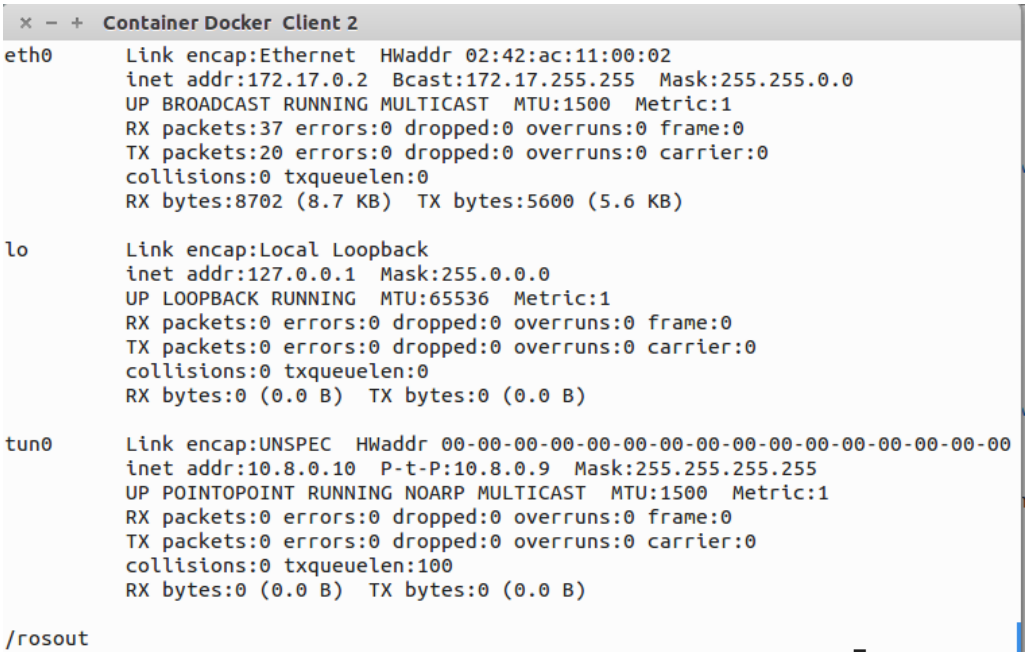


Figure 4.5: Client 2 Docker Container Output

When the Client 2 Docker Container will be run, the output in Figure 4.5 will be obtained.

The outputs obtained, in the Figures 4.4, 4.5 and 4.3, demonstrate that the containers are connected to the same tun0 network, and are able to interact inside a ROS environment, since all of them can see the */rosout* topic.

Chapter 5

PARLOMA

This chapter provides an overview of the PARLOMA project. After the theoretical presentation on Service Robotics and Cloud Robotics, a generic Multi-Machine VPN-based ROS Network has been developed. Now, the focus will be moved to the specific appliance of this solution: The PARLOMA project.
This thesis provides the project with significant features to provide real remote communication, out from a local area network (LAN). Therefore, an overview of the current state of the art of PARLOMA is necessary to understand some subsequent considerations.

This chapter is organized in this way: Section 5.1 provides a thorough understanding of the PARLOMA principles of work, then Section 5.2 illustrates the open issues and the related contributions proposed by the Author. Finally, Section 5.3 contains the implementation of a VPN-based ROS Network, directly on the PARLOMA project. To gather information on PARLOMA, L. Russo has been often consulted, together with his PHD thesis works [20], the PARLOMA website [1] and [19].

5.1 State-of-the-art

The PARLOMA project was born from the collaboration between Politecnico di Milano and Politecnico di Torino, thanks to the Alta Scuola Politecnica (ASP). It was started by a team of students from the VIII ASP cycle, about six years ago, and it is still in development under the sponsorship of the Ministry of Education, University and Research (MIUR). It is developed by L. Russo, G. Airò Farulla, and others colleagues, under the supervision of Professors P. Prinetto and B. Bona.

As the PARLOMA website[1] states: "The PARLOMA project is aimed at creating a robotic system to allow remote communication between two deafblind people, a deafblind person to a deaf person and a deafblind person to a hearing person with a knowledge of sign language". In order to ensure this communication, the project is based on anthropomorphic robotic hands, which reproduce the gestures of Sign Language.

[1] PARLOMA website, https://parloma.github.io/

Practically, the project aims to offer deafblind people the opportunity to communicate remotely, since, at the moment, no device provides this solution. Deaf people communicate through the SL (sign language) since they are able to see, instead, deafblind people use TLS (tactile sign language). Although these ways of communicating allow them to converse effectively, it has enormous limitations. In particular, the main problems are the following:

- Both the interlocutors (*receiver* and *signer*) must be in the same position when they communicate, as they must keep their hands in contact;
- Only one-to-one communication is possible, as the receiver needs to maintain contact with the signer's hand.

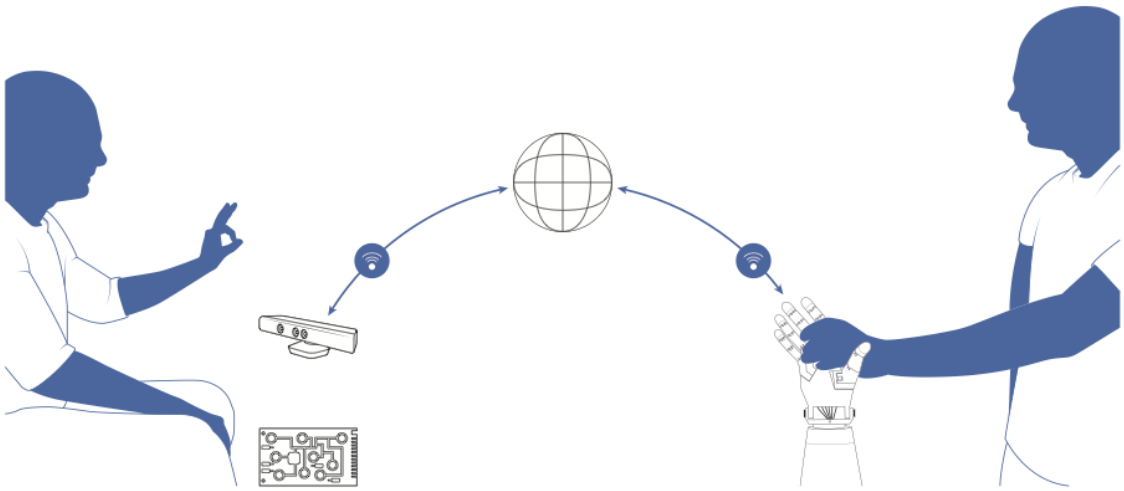


Figure 5.1: How PARLOMA works.

To overcome these limitations, PARLOMA would allow both remote communication, thanks to the use of a cloud robotics platform, and multiple conversations, by providing to multiple receivers to remotely replying the signer gestures. Figure 5.1 illustrates how PARLOMA operates.

PARLOMA is based on the three main technologies listed below:

1. *Robotic operating system (ROS)*, used as environment for managing communication between devices.
2. *Amazon Web Service (AWS)*, used to host the cloud computing platform.
3. *Raspberry Pi and Arduino Mega*, used as a system processing unit, connected with the PARLOMA hand. When it receives commands from the cloud platform, it controls accordingly the robotic hand.

Moreover, the tasks performed by PARLOMA can be divided into three main groups:

- *Hand tracking*, performed by means of the Kinect device or a Leap Motion sensor (see Figure 5.2).
- *Transmission of data*, between the signer and the receiver, which must be guaranteed as secure and private.
- *Gestures reproduction* using a robotic hand that moves according to the signs accomplished by the signer. (see Figure 5.3).

Figure 5.2: Hand tracking sensor devices.

Among these groups, the solution developed by the Author involves the *data transmission*, where a secure channel is required to forward information from the signer to the receiver devices. For this scope, in the next section the architecture designed to guarantee the success of the communication between devices will be discussed, and the Author's contribution will be addressed.

Figure 5.3: Robotic hand and Control Box containing Raspberry and Arduino

5.1.1 Transmission Architecture

For reasons of simplicity, the architecture will be divided into two parts, explained below:

- **Communication Architecture,** which describes the global communication between input and output devices.
- **Cloud Interface Architecture,** which illustrates in detail how the behaviours necessary to ensure the communication are implemented.

Communication Architecture. This architecture ensures the data streaming between devices, from the input to the output, through the cloud platform. Thus, it is possible to assume that input device, remote server and output device are already able to reach each other, being connected in a common network. Figure 5.4 shows the logic diagram of the Communication Architecture, that is composed of three modules as follows:

1. The **Input module**, on the left side, acquires the input signals of the signer device (Figure 5.2) in real-time and processes them in order to get the hand tracking. This module incorporates the input driver and the ROS driver, the first represents the actual sensor, the second, instead, converts the actual measurements in ROS messages. Then the ROS driver send these messages to the following unit.

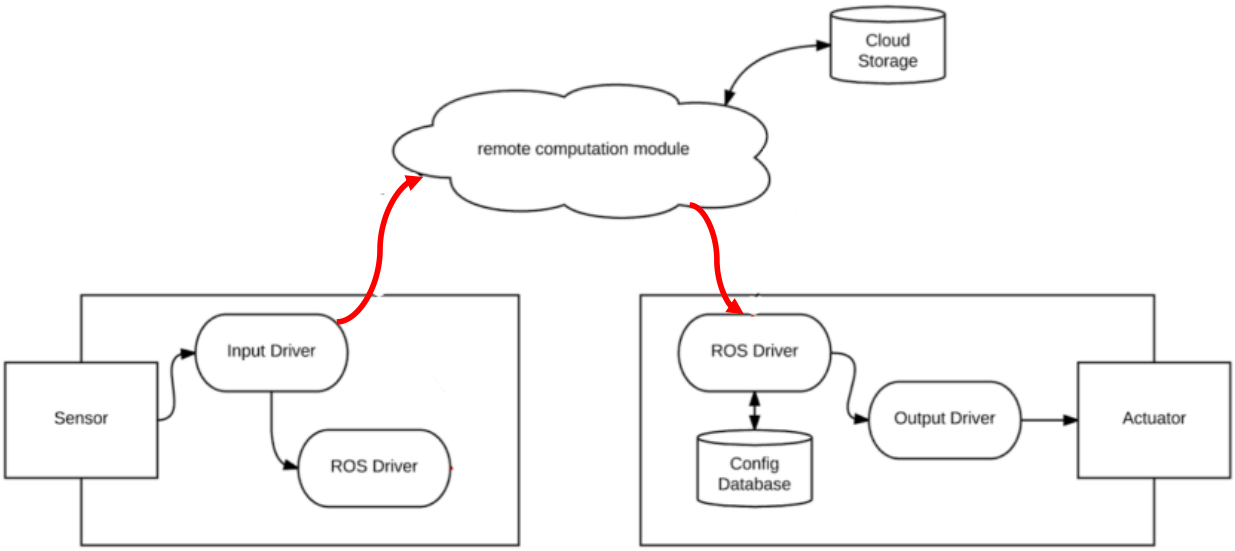


Figure 5.4: Communication Architecture Logic Diagram

2. The **Remote Computation Module** guarantees the communication between Input and Output modules through an ad-hoc VPN channel.
3. The **Output module**, on the right side, receives the hand tracking data from the previous module through ROS messages and, complementary to the input module, sends commands to the actuator device passing through the ROS driver and the output driver (Figure 5.3).

Cloud Interface Architecture. This architecture is responsible for starting up the communication network needed to accomplish the data transmission described above. The cloud platform contacts all the connected devices, in order to establish independent communication channels when needed.

The Cloud Interface Architecture is shown in Figure 5.5, where it is possible to distinguish three main components:

- The **Web GUI** used by users to access the platform and configure their I/O devices.
- **Input Modules Manager** and **Output Modules Manager**, needed to communicate with input and output devices. This communication is insured thanks to the CI (Cloud Interface) module, which is continually running inside the connected I/O modules.

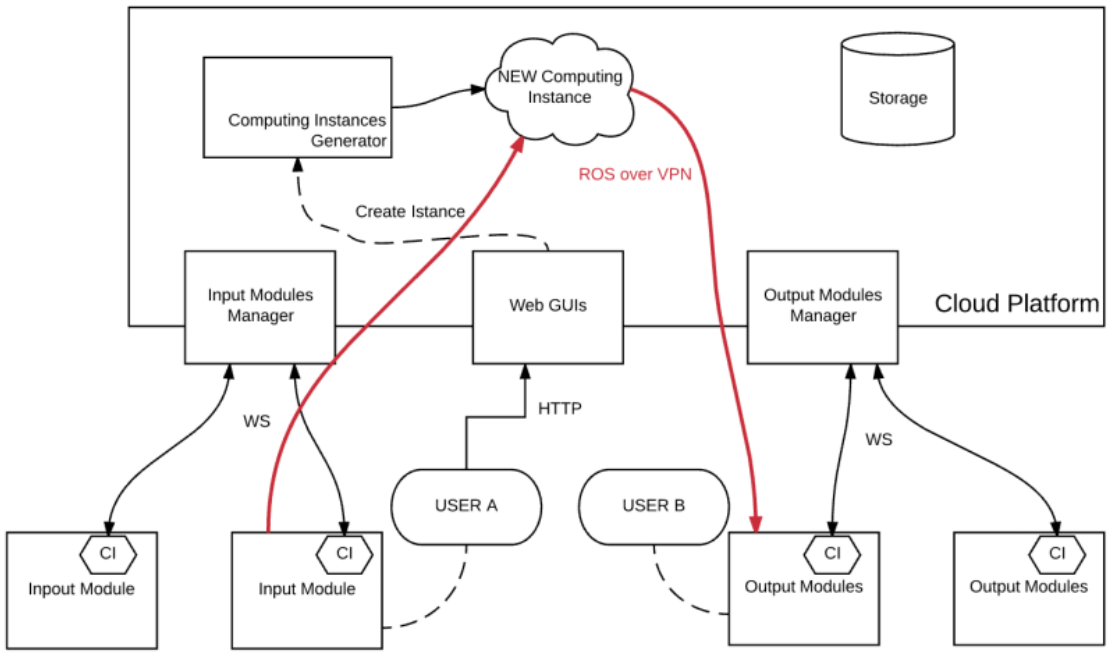


Figure 5.5: Cloud Interface Architecture

5.2 Open Issues and Author's contribution

Referring to Figure 5.5, it is possible to notice a red line that connects the *Computing Instance* with the *Input and the Output Modules*. This line, identified as *ROS over VPN*, indicates the presence of a Multi-Machine VPN-based ROS Network that allows to launch an extended ROS network including both the input/output devices and the server. At the same time, a process is needed in order to automate the initialitation of different *New Computing Instances*. Until today, this possibility was not present and for this purpose, the work carried out by the Author has been used.

The PARLOMA architecture is similar to the architecture of a teleconferencing software such as *Skype*. To understand better how the developed solution has been implemented, referring to Figure 5.6, the general steps of a call on PARLOMA will be explained.

A generic user A, through the Web GUI, initiates a call with another user B. After that the cloud platform verifies that the corresponding I/O devices are available and connected, it will notify the incoming call to user B. When the call is accepted, the platform generates a Docker container, through the commands *docker build . -t parloma:server* and *docker-compose up*. It will represent the *NEW Computing Instance*. Within this container the *VPN server* will be automatically executed. Next, the involved Input/Output Modules will be informed by the Cloud Platform about the *Address: xxx.xxx.xxx.xxx* of the VPN server and two different *Docker containers* will be built and executed through the commands:

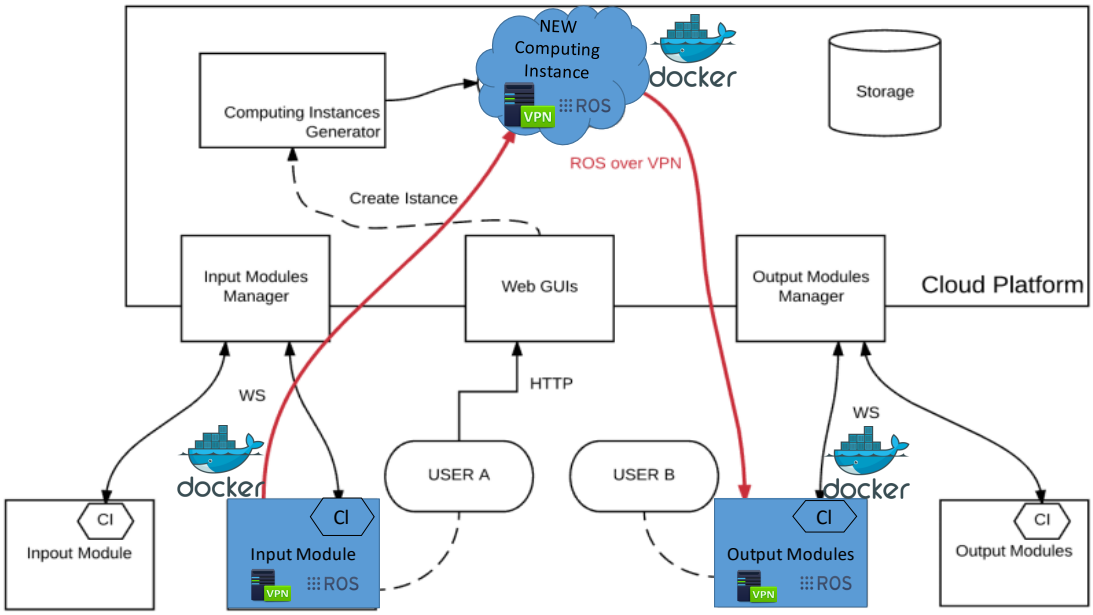


Figure 5.6: Architecture Solution

- *docker build . -t parloma:client_input*
- *docker-compose run -e IP=xxx.xxx.xxx.xxx client_input.*
- *docker build . -t parloma:client_output*
- *docker-compose run -e IP=xxx.xxx.xxx.xxx client_output*

Finally, both these containers will initiate two separate *VPN client instances* that will connect to the *NEW Computing Instance container* using the *Address* parameter. All the involved Docker containers will run different ROS nodes, depending on the tasks they have to perform. In particular:

- the *parloma:client_input container* executes the *Publisher* node, that publishes the *Hand Tracking* measurements, on the */output/serial_topic* topic.
- the *parloma:server container* executes two nodes: the *ROS master* node, that manages the communication between the other ROS nodes, and a *Computation* node that translates the information read from the topic *output/serial_topic* in commands that will be published on the */joint _command* topic.
- the *parloma:client_output container* executes the *Subscriber* node, that reads the measurements from the */joint_command topic* and performs the *Gestures reproduction.*

This mechanism had not yet been automated, and this is the main point of this thesis development: to automate this interfacing process using Docker Container and VPN, allowing ROS environment over the Internet. In the next section will be explained the development in details.

5.3 PARLOMA Contribution

In the previous Chapter 4, the way to configure a ROS network over Internet was described. In this part instead how to use this network in order to improve the PARLOMA project will be explained.

Figure 5.7 shows the directories and files that compose the project.

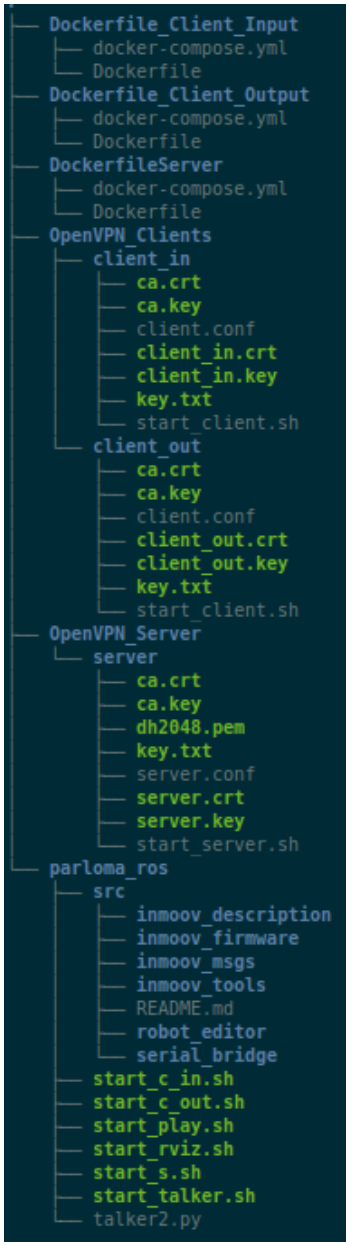


Figure 5.7: PARLOMA Directory

VPN Configuration

The VPN configuration corresponds with the one explained in subsection 4.2. The three configuration files produced are:

- A configuration file, as client1, for a general *Input module*.
- A configuration file, as client2, for a general *Output module*.
- A configuration file, as server, for a general *New Computing Instance*.

Since both the clients are tailor-made for the same VPN server, they are really similar and correspond with the subsection 4.2. Just the name of the certification and key files are different:

- client1.key, client1.crt -> client_in.key, client_in.crt
- client2.key, client2.crt -> client_out.key, client_out.crt

These files have to be substituted also in the client.conf file listed in 4.2.

DOCKER and ROS Configuration

After that the VPN files have been properly built, the next step is to set up the ROS environment. In order to move the computing part away from the Input/Output modules, the ROS Master has been executed into the VPN Server. The latter, in turn, is located, referring to PARLOMA and in particular to the Cloud Interface (see Figure 5.6), into the Cloud Platform and a new VPN Server instance corresponds to each New Computing Instance. In particular, these servers are initiated inside docker containers that run in a Amazon Web Services (AWS) platform.

Right after each call is initiated within a PARLOMA system, three Dockerfile will be built. In particular, referring to the Figure 5.6, the following Docker container will be started:

- A client, which represents the Input Module Instance
- A second client, which represents the Output Module Instance
- A server, which represents the New Computing Instance

The two client containers will obtain the server address from the I/O Modules Manager and will be able to connect with it.

Like for the general Multi-Machine VPN-based ROS Network, now the configuration needed to start Docker containers (both Clients and Server) will be discussed.

Server Dockerfile

The Dockerfile for the server container is shown in Listing 5.1

```
1  FROM osrf/ros:indigo-desktop-full
2
3  RUN \
4   svn checkout https://github.com/creos92/parloma.git/trunk/Server &&\
5   svn checkout https://github.com/creos92/parloma.git/trunk/parloma_ros &&\
6   apt-get update &&\
7   apt-get install build-essential checkinstall -y &&\
8   apt-get install libssl-dev liblzo2-dev libpam0g-dev wget -y &&\
9   wget https://swupdate.openvpn.org/community/releases/openvpn-2.4.4.tar.gz &&\
10  sudo tar zxvf openvpn-2.4.4.tar.gz &&\
11  cd openvpn-2.4.4 && ./configure && make && checkinstall -y &&\
12  /bin/bash -c 'source /opt/ros/indigo/setup.bash && \
13  cd /parloma_ros && catkin_make'
14
15 CMD cd parloma_ros && chmod +rx start_s.sh && sync && ./start_s.sh
```

Listing 5.1: PARLOMA Server Dockerfile

This listing corresponds to the one of a general Server Dockerfile 4.3. For this reason only the different commands will been explained:

- Command at line 5 downloads the directory that contains the ROS packages needed for this particular application. It could contain many programs that consequentially start different nodes inside the ROS Multi-Machine Network.
- Commands at line 12-13 make the *ros_parloma packages*. However, the ROS environment must be firstly sourced, in fact the source command evaluates the file setup.bash and executes it in the terminal window, into which the ros_parloma packages will be made.
- Command at line 15 initiates the bash file start_s.sh (see Listing 5.5) that automates the process of creating a VPN server and of implementing of ROS.

Then, through the *docker build . -t parloma:server* command, the Dockerfile will be compiled resulting in a new image. In order to start a container from this obtained image, the command *docker-compose up* will be executed. The container will be initiated starting from the configuration options listed in the *docker-compose.yml* (see Listing 5.2).

```
1  parloma_server:
2    image: parloma:server
3    stdin_open: true
4    tty: true
5    privileged: true
6    ports:
7      - 1194:1194
```

Listing 5.2: PARLOMA Server docker-compose.yml file.

- Command at line 1 specifies the tag that will be assumed by the Docker Container.
- Command at line 2, instead, contains the image that will be used to start the container.
- Commands at lines 3-4-5 are useful when interactign with the started container is needed.
- Commands at lines 6-7 publish the container's ports to the host and viceversa. In this way, it is possible to use the services hosted by that port. In this case the port corresponds to 1194.

Client Dockerfile

The Dockerfile for the *Client Input Module* container is shown in Listing 5.3; instead, the one for the *Output Module* is not listed, since it is almost similar. It is possible to notice also that many commands correspond with the ones of the Server Dockerfile 5.1.

The main differences are explained below:

- Command at line 4 provides the download of the configuration and certification files for the clients described in the beginning of this Chapter.
- Command at line 15 initiates the bash file start_c_1.sh (see Listing 5.6), that automates the process of creating a VPN client and of implementing of ROS.

```
1  FROM osrf/ros:indigo-desktop-full
2
3  RUN \
4   svn checkout https://github.com/creos92/parloma.git/trunk/Client &&\
5   svn checkout https://github.com/creos92/parloma.git/trunk/parloma_ros &&\
6   apt-get update &&\
7   apt-get install build-essential checkinstall -y &&\
8   apt-get install libssl-dev liblzo2-dev libpam0g-dev wget -y &&\
9   wget https://swupdate.openvpn.org/community/releases/openvpn-2.4.4.tar.gz &&\
10  sudo tar zxvf openvpn-2.4.4.tar.gz &&\
11  cd openvpn-2.4.4 && ./configure && make && checkinstall -y &&\
12 /bin/bash -c 'source /opt/ros/indigo/setup.bash && \
13  cd /parloma_ros && catkin_make'
14
15 CMD cd parloma_ros && chmod +rx start_c_in.sh && sync && ./start_c_in.sh
```

Listing 5.3: PARLOMA Client Input Module Dockerfile, the one for the *Output Module* is almost similar, just replace start_c_in with start_c_out

Through the *docker build . -t parloma:client_in* command the Dockerfile will be compiled resulting in a new image. In order to start a container from this obtained image, the command *docker-compose run -e IP=ServerAddressIP parloma:client* will be executed. The IP is the environment parameter that will be obtained from the Cloud Platform. It contains the *IP Address* of the Server instance to connect with. This IP address will be substituted to the "ipserveraddress" in the *client.conf* file. The container will be initiated starting from the option configuration listed in the *docker-compose.yml* (see Listing 5.4).

```
1  parloma_client_in:
2    image: parloma:client_in
3    stdin_open: true
4    tty: true
5    privileged: true
6    volumes:
7      - /tmp/.X11-unix:/tmp/.X11-unix:rw
8    environment:
9      - DISPLAY=\$DISPLAY
10     - QT_X11_NO_MITSHM=1
```

Listing 5.4: PARLOMA Client Input Module docker-compose.yml file, the one for the *Output Module* is almost similar, just replace client_in with client_out

Almost all the lines correspond with the docker-compose.yml file of the server

- The container tag and of the image name are different.
- The added lines, from 6 to 10, are needed to start a GUI. Practically they enable the use of graphical user interfaces within Docker containers, in such a way that graphical application such as Rviz can be executed properly.

Automated script

The main bash files will be listed in order to understand which commands are used to implement the architecture. These files are the ones cited at the end of clients/server Dockerfiles 5.3 and 5.1 .

Server initiation Bash File

```
1  #!/bin/bash
2  source ./devel/setup.bash
3  chmod +rx ./start_talker.sh &&
4  chmod +rx /Server/certificati/server/start_server.sh &&
5  cd /Server/certificati/server && sync && ./start_server.sh &
6  sleep 1s
7  roscore &
8  sleep 2s
9  export ROS_IP=10.8.0.1
10 export ROS_MASTER_URI=http://10.8.0.1:11311
11 cd /parloma_ros && ./start_talker.sh
```

Listing 5.5: PARLOMA start_s.sh bash file.

The commands similar to the Listing 4.7 will not be discussed:

- Line 7 starts a collection of nodes that are pre-requisites of a ROS-based environment. The roscore command corresponds with the initiation of the ROS MASTER. This allows the communication between different ROS nodes.
- Line 11 represents the command that starts the *Talker node*. In this case, this node is subscribed to the topic /output/serial_topic that collects information from Input Sensor and publishes on the topic joint_command that contains the command needed to move the robotic hand.

Client Input Module Bash File

```
1  #!/bin/bash
2  source ./devel/setup.bash
3  sed -i "s/indirizzoipserver/"$IP"/g" ../Client/client_in/client.conf
4  chmod +rx start_play.sh /Client/client_in/start_client.sh &&
5  cd /Client/client_in && sync && ./start_client.sh &
6  sleep 3s
7  export ROS_IP=$(/sbin/ifconfig | grep "inet addr:10.8"
8      |awk -F: '{print $2}' | awk '{print $1}')
9  export ROS_MASTER_URI=http://10.8.0.1:11311
10 cd /parloma_ros && ./start_play.sh
```

Listing 5.6: PARLOMA start_c_in.sh bash file.

The start_c_in.sh is very similar to the start_server.sh file 5.5. For this reason only the different statements will be illustrated:

- Command at line 7 extracts the IP address that starts with 10.8.xxx.xxx, resulting by the command *ifconfig* executed inside the container. It uses this IP to uniquely identify the own machine on the ROS environment.
- Command at line 10 starts a node that publishes on /output/serial_topic the data read from the sensor module.

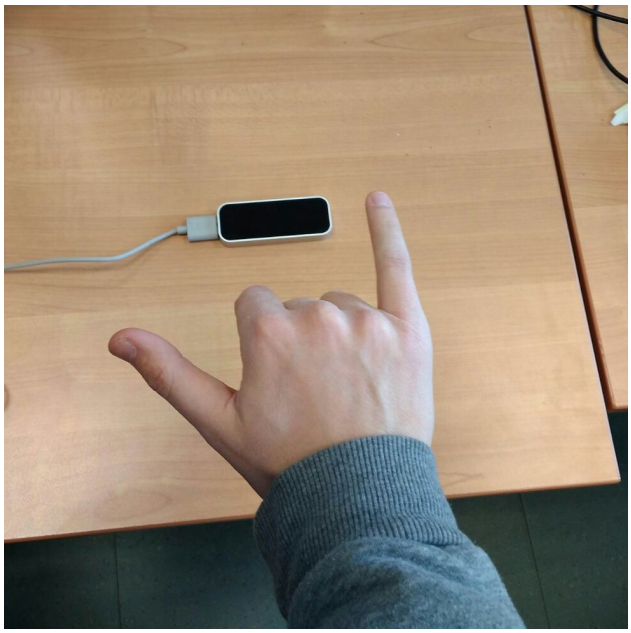


Figure 5.8: Leap Motion measurements acquisition

When the Client Input Module Docker Container will be started, the ROS Node will start to read the sensor module measurements and will publish it on /output/serial_topic (see Figure 5.8).

Client Output Module Bash File

As fa as the Output side is concerned, the start_c_out.sh file is almost similar to the start_c_in.sh.

```
1  #!/bin/bash
2  source ./devel/setup.bash
3  sed -i "s/indirizzoipserver/"$IP"/g" ../Client/client_out/client.conf
4  chmod +rx start_rviz.sh /Client/client_out/start_client.sh &&
5  cd /Client/client_out && sync && ./start_client.sh &
6  sleep 3s
7  export ROS_IP=$(/sbin/ifconfig | grep "inet addr:10.8"
8          |awk -F: '{print $2}' | awk '{print $1}')
9  export ROS_MASTER_URI=http://10.8.0.1:11311
10 cd /parloma_ros && ./start_rviz.sh
```

Listing 5.7: PARLOMA start_c_out.sh bash file.

The few different statements are the following:

- Lines 3-4-5 are similar (just substituting client_in with client_out).
- The last command starts a different node from the Client Input side. In this case, this node publishes on joint_command allowing to see on Rviz the Robotic Hand virtualization (see Figure 5.9). Obviously, also the Robot Hand control unit is subscribing on this topic, allowing the robotic hand motion (see Figure 5.10).

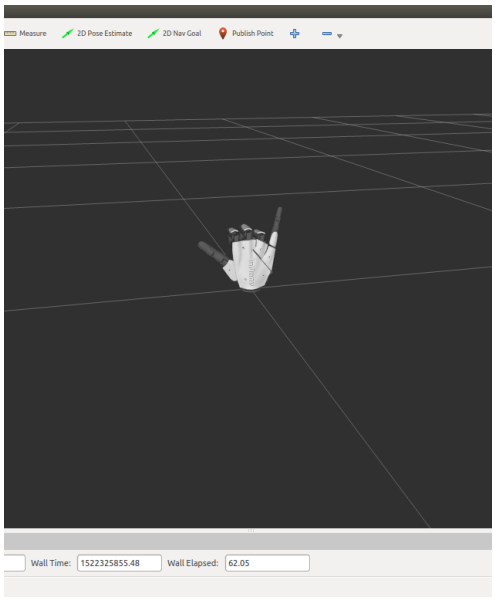


Figure 5.9: Rviz Visualization

Figure 5.10: Parloma Robotic Hand Moving

Chapter 6

Experiments

After the VPN-based ROS network description and the practical application on PARLOMA, some experiments are surely useful to discuss the main characteristics of the developed solution. In particular, the main parameters are the following:

- The **Network latency** that represents the amount of time taken by a data packet to flow from one point to another through the network.
- The **Network throughput error** that represents the error between the rate at which packets are sent and the rate at which they are received over the network.

To perform a profitable test, three ROS networks were compared. They are all made up of the same ROS nodes:

- A **Master** node on a Server Docker Container
- A **Publisher** node on a Client(1) Docker Container
- A **Subscriber** node on a Client(2) Docker Container

The involved networks, instead, are as follows:

- **Single Machine VPN-based ROS network**: the three Docker containers run on a single machine.
- **Local Multi-Machine VPN-based ROS network**: the three Docker containers run on three different machines connected to the same LAN.
- **Multi-Machine VPN-based ROS network**: the three Docker containers run on three different machines connected through the internet from different LAN.

As for the data packets sent by the Publisher node, they consist of two fields:

- A **string** of variable size
- A **time-stamp** that indicates the time at which data have been sent (in second.nanosecond format)

To reach a reasonable amount of measurements, sets of 500 packets were sent in the three different networks, varying the size of the string field, i.e. the number of characters. Each of these packets contains 10 messages that have the same dimension, and increase progressively (10 messages of dimension 0, 10 messages of dimension 1, and so on, until 10 messages of dimension 500).

In order to collect data, the Publisher node sends the data packets while a Subscriber node stores these packets. Successively the data are processed to obtain the desired information about throughput errors and latency. In next subsections, these two parameters will be discussed separately and some conclusions will be addressed.

6.1 Latency

In order to measure the latency of the three networks, the function rospy.Time.now() was used. It returns the float seconds of the current time.
Shortly, the mechanism for calculating latency works in this way: the Publisher saves the current time in the *timestamp* field of a packet, using the function rospy.Time.now(), together with the *string* composed of the n characters (n increases progressively, every 10 packet sent); therefore, once the subscriber receives the packet, it calculates the difference between the current result of the rospy.Time.now() and the time value of the received packet to obtain the latency error.

Several factors can influence the latency like:

- The queuing latency on the listening machine.
- The clock offset between the involved machines in the ROS network.
- The network latency of the local or wide area network.

Nevertheless, since the same machines were involved in the three configurations and since the measurements were performed under the same conditions, the experiments can be considered reasonable for the evaluation of network latency. Figure 6.1, 6.2 and 6.2 show the results from the three network configurations.

As expected, the Multi-Machine VPN-based ROS network (Fig. 6.3) is the one that shows the highest latency peaks and also the highest average latency. The decreasing trend in the first 100 sets of packets, in Multi-Machine VPN-based ROS network figure 6.3, can be motivated by the loading of the communication tunnel through the VPN, which takes some time to setting a stable channel.

Regarding instead the other two configurations, the LAN has a similar behaviour respect to the WAN; on the other hand the Single Machine VPN-based ROS network is the one that shows the smallest latency, which can be considered almost negligible. The latter has not a growing trend respect to the local and wide area network, in which the latency depends on the size of the packets.

All the other peaks that appear in all the Latency test are caused by the neck of the bottle due to the enormous amount of data. Once the bottle neck was resolved, the latency returned as constant as before.

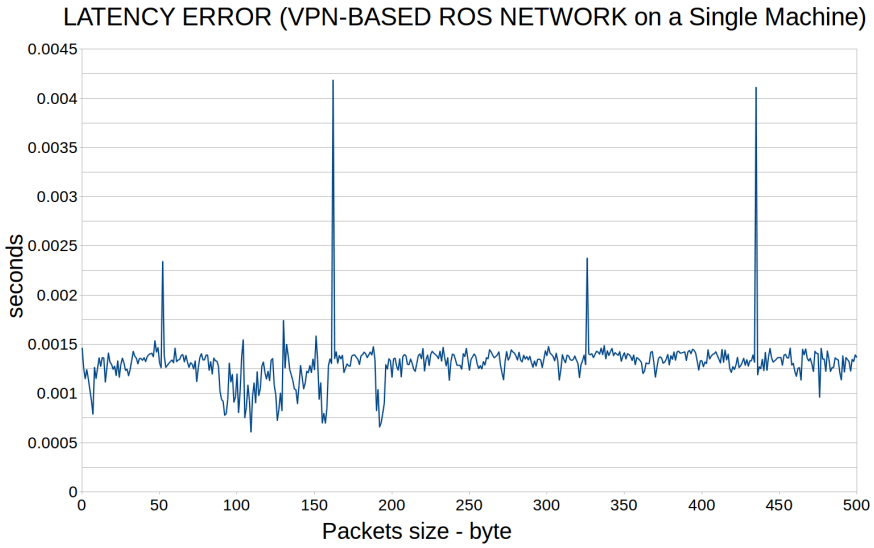


Figure 6.1: Latency error of a VPN-based ROS network on a Single Machine

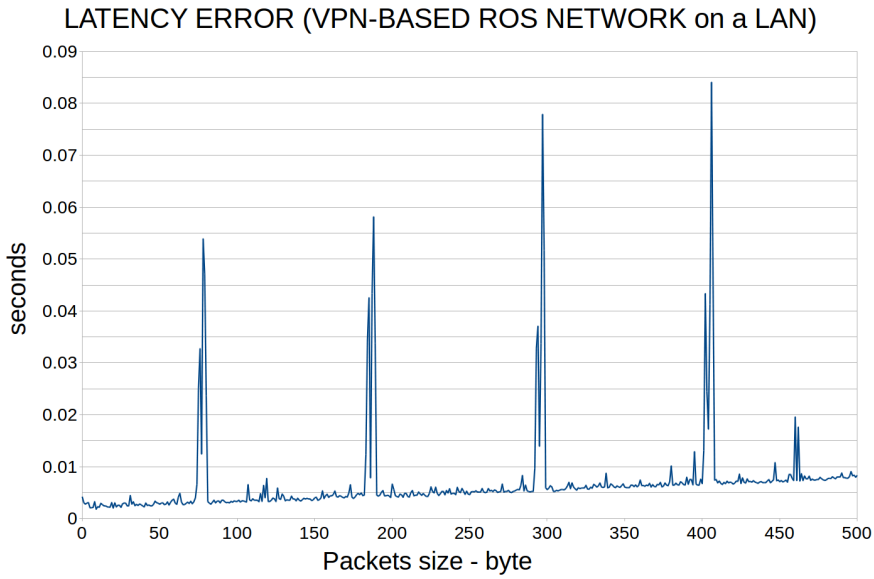


Figure 6.2: Latency error of a Multi-Machine VPN-based ROS network on a LAN

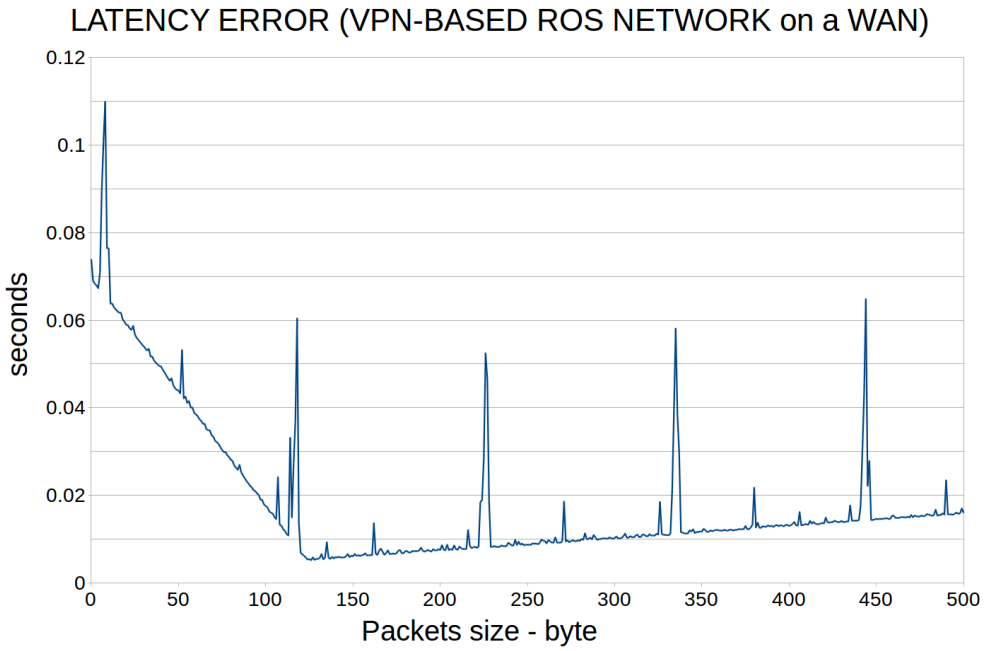


Figure 6.3: Latency error of a Multi-Machine VPN-based ROS network on a WAN

6.2 Throughput error

In order to estimate the throughput error in the three network configurations, the call rospy.Rate() was used. This function imposes a rate of the communication loop.
The Subscriber and the Publisher operate in this way: the Publisher sets a rate of communication corresponding to 10 Hz, using the rospy.Rate (10) instruction, and then starts sending the data packets while the Subscriber stores them. Next, the difference between successive packets arriving times was calculated and compared with the difference between successive packets sending. Then the error between them was used to indicate the accuracy of throughput.

Figures 6.4, 6.5 and 6.6 show the results of the three networks.

Once again, the Multi-Machine VPN-based ROS network is the one that shows the worst results even in terms of throughput error. The tendency is similar to the latency error, since the general behaviour of the Single Machine ROS network VPN-based is better than the other two networks, less than some fairly high peaks.

The measures result more fluctuating than those ones of latency. In conclusion, it should be noted that the same peaks shown by the Multi-Machine ROS network VPN-based in terms of latency are also present in the throughput errors graphs, corresponding to the same values.

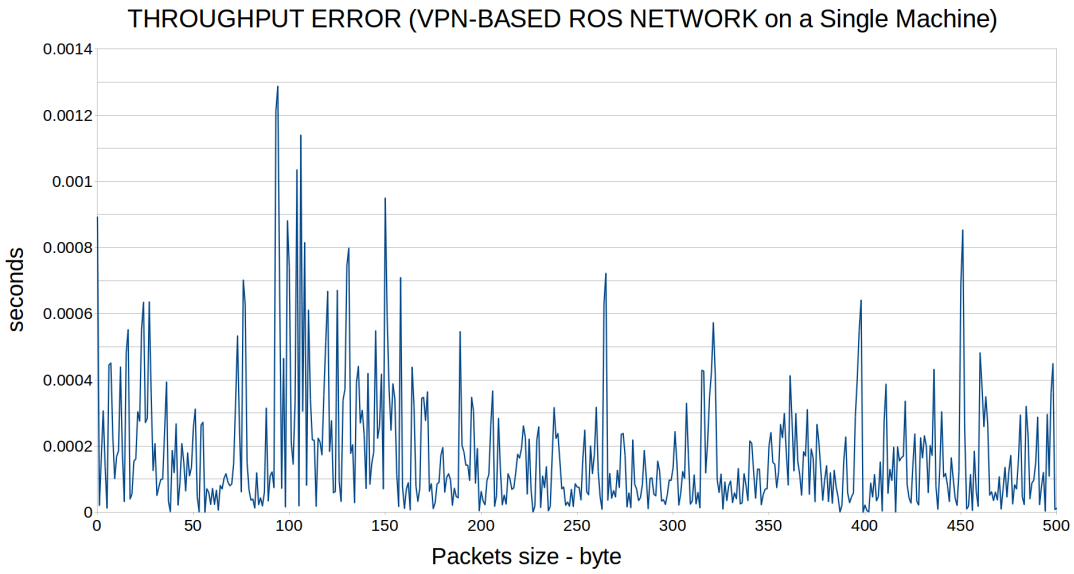


Figure 6.4: Throughput error of a VPN-based ROS network on a Single Machine

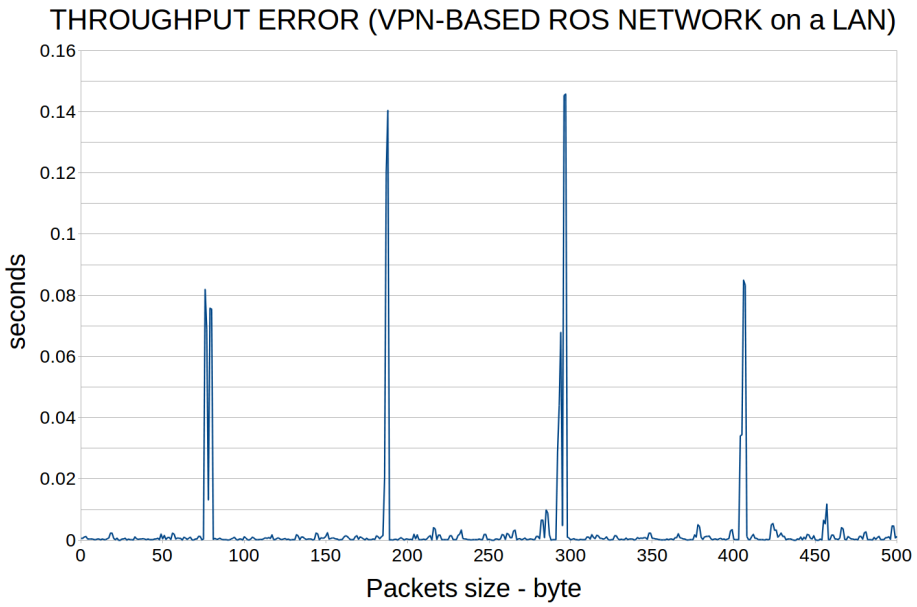


Figure 6.5: Throughput error of a Multi-Machine VPN-based ROS network on a LAN

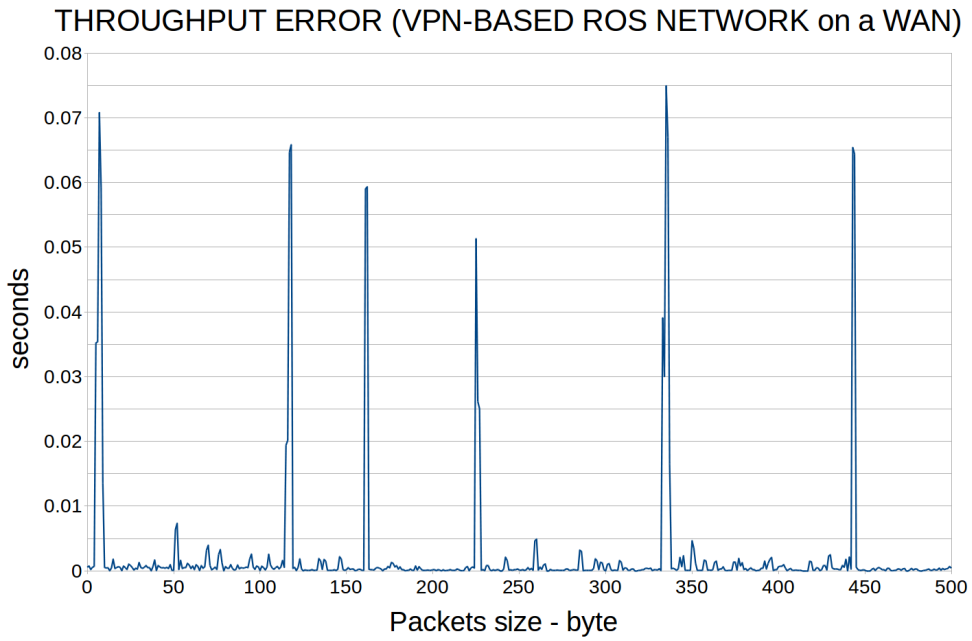


Figure 6.6: Throughput error of a Multi-Machine VPN-based ROS network on a WAN

6.3 Results

Some conclusive remarks are illustrated in order to further discuss about the obtained results.

- The developed architecture is applicable in several applications, even if it does not observe the characteristics of Hard Real-Time systems, i.e., those systems where missing a deadline compromises the failure of the system. Indeed, although the behaviour is positive, some peaks may compromise the deadline meeting. Anyhow, this does not correspond to an actual issue since hard real-time system can never be realised adopting distributed networks through Internet. The last consideration comes from the fact that the Internet behaviour is not predictable.
- In the first graph (the latency one) the performances of the VPN-based ROS Network are a bit worse than the ones of the local network or single computer. This represents an excellent result because the packets sent within the VPN must go across a much longer path than the one sent inside a local network.
- In the second graph (the one related to the throughput error) the VPN actions are much more unstable. Probably, it is caused by the traffic congestion generated in the VPN channel. In fact, with the exception of the first peak generated by the initialisation of the network, the general tendency is composed of "triangular waves", in which a large peak is surrounded by smaller ones.

Chapter 7

Conclusions

This Thesis presents the theoretical study and an application project on the theme of Cloud Robotics. As far as the application part is concerned, it focuses on a real case study: important contributions to some open issues of the PARLOMA project were given in order to improve its cloud platform and network architecture. Then, concrete experiments were provided for testing the after-mentioned improvements.

This Chapter provides the main results reached by the Author's work and the potential developments that can be carried on. In the first part of the Chapter the final remarks are discussed. Next, in Section 7.2, the future developments are proposed.

7.1 Final Remarks

The architecture built succeeded in implementing external ROS networks, connected to different LANs. Moreover, the quality of the results has been proved with the PARLOMA application that is making usage of the extended ROS networks developed.

Finally, this allows to deafblind people, that are located everywhere around the world, to communicate remotely with their interlocutors thanks to PARLOMA.

In fact, until now, the PARLOMA parts could communicate just on local ROS networks. This meant that the Cloud Platform and all the I/O devices had to be connected to the same LAN to be identified by the ROS Master. Obviously this limitation strongly affected the potential of the system making it almost useless, since it was designed to operate remotely. Thanks to the Author's contribution, these limitations have been overcome, allowing the expansion of the ROS architecture over the Internet.

Thus, the objective of the thesis work has been completely achieved, even if it is currently a prototype not ready to be commercialised yet. Besides some interesting outcomes are illustrated, along with some tips (described in Section 7.2) aimed at moving the application from an experimental status to a commercial one.

The main aspects that deserve to be highlighted are:

- The network is flexible and highly scalable. Thanks to *Docker containers*, it can be used by many platforms without any problem. It easily interfaces to the host system, providing

always the same functionalities.

- The set up is extremely simple. In fact, many instances of the server and client can be easily initiated using very few commands.
- The robustness of the architecture is guaranteed regardless of the amount of connected devices.
- The work has been implemented with the PARLOMA cloud platform. In fact, once the Docker images was built and supplied with a VPN-based ROS environment, it was successfully integrated with the PARLOMA application.

This Thesis work represented a precious experience for the Author. Most of the goals described before begin of the work were achieved and will be of interest for the future of HotBlack Robotics company and, surely, of PARLOMA project.

7.2 Future developments

Even though the results achieved are good and match the requirements, some further effort can be spent to improve the project. The following suggestions are for those intentioned to enhance the performances of the work done:

- The procedure to produce the *CA (Certification Autority)* files, used to implement a unique and secure VPN solution, should be automated, in order to manage many *client/server VPN* solutions. At the moment, although many docker containers (and, thus, many *New Computing instances*) can be started, they uses the same certificates. This last fact could obviously compromise the security of the system. A best scenario, instead, would be when different I/O modules request access to the platform and each of these couples have their own certificates.
- Security issues have not been solved during application development. AWS protections have been disabled with the aim of not interfering with the interfacing process. Nevertheless, before integrating the service with the PARLOMA platform, a deeper analysis would be needed to understand which security defences are needed for providing a secure environment for users.
- As far as the application of PARLOMA is concerned, a graphical interface in the platform should be integrated with the goal of guaranteeing a user-friendly experience. While, at the moment, the Docker installation and Docker containers initiation must be lunched through command line terminal.

Bibliography

[1] L. Russo et colleagues. *PARLOMA, a telecommunication system for deafblind people*. URL: https://parloma.github.io/.

[2] A.Kristoffersson; S. Coradeschi and A. Loutfi. "A Review of Mobile Robotic Telepresence". In: (2012).

[3] E. F. Crist and J. J. Keijser. *Mastering OpenVPN*. PACKT Publishing, 2015.

[4] *Amazon Elastic Computing Cloud*. URL: https://aws.amazon.com/it/ec2/.

[5] M. Quigley; B. Gerkeyy; K. Conleyy; J. Fausty and T. Footey. "ROS: an open-source Robot Operating System". In: *Computer Science Department, Stanford University, Stanford, CA* (2009).

[6] *FlexiScale Cloud Comp and Hosting*. URL: www.flexiscale.com.

[7] GoGrid. *Cloud Hosting, CLoud Computing and Hybrid Infrastructure from GoGrid*. URL: http://www.gogrid.com.

[8] B. Kehoe; A. Matsukawa; S. Candido; J. Kuffner; K. Goldberg. "Cloud-based robot grasping with the google object recognition engine". In: *Technical report, IEEE International Conference on Robotics and Automation (ICRA)* (2013).

[9] K. Goldberg. *Understanding no robot is an island. Humans Invent*. URL: goldberg.berkeley.edu/cloud-robotics.

[10] Google. *Google App Engine*. URL: http://code.google.com/appengine.

[11] Rackspace Hosting. *Dedicated Server, Managed Hosting, Web Hosting by Rackspace Hosting*. URL: http://www.rackspace.com.

[12] Todd Humphreys. "Cockrell school researchers demonstrate first successful 'spoofing' of uavs." In: *University of Austin, Texas Magazine* (2012).

[13] *RobotShop, Pioneers Cloud Robotics, Interview With Mario Tremblay*. 2011. URL: www.robotshop.com/blog/en/myrobots-com-pioneers-cloud-robotics-interview-with-mario-tremblay-1348.

[14] J. Kuffner. "Cloud-Enabled Robots. In proc. of the IEEE Intl. Conf. on Humanoid Robots". In: 2010.

[15] Microsoft. *Microsoft Azure*. URL: www.microsoft.com/azure.

[16] T.Grance P.Mell. "The NIST and Definition of Cloud and Computing". In: 2011.

[17] International Federation of Robotics (IFR). "Survey on Service Robotics". In: (2016).

[18] World Robotics. "Executive Summary on Service Robots". In: (2016).

[19] L. Russo. "Progetto di Innovazione Sociale, SIN00132 PARLOMA". In: *Smart Cities and Communities and Social Innovation* (2017).

[20] L. O. Russo. "Leveraging the Cloud to develop Service Robotics Applications". MA thesis. Politecnico di Torino, 2017.

[21] Salesforce. *Salesforce CRM*. URL: http://www.salesforce.com/platform.

[22] SAP. *SAP Business ByDesign*. URL: https://www.sap.com/products/business-bydesign.html.

[23] Dezhen Song, Ken Goldberg, and Nak Young Chong. "Networked Telerobots". In: *Springer Handbook of Robotics*. Ed. by Bruno Siciliano and Oussama Khatib. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 759–771. ISBN: 978-3-540-30301-5. DOI: 10.1007/978-3-540-30301-5_33. URL: https://doi.org/10.1007/978-3-540-30301-5_33.

[24] B. Sosinsky. *Cloud Computing Bible.* Wiley, 2011.

[25] M. Beetz; J. M. M. Montiel; M. Tenorth and O. Zweigle. "RoboEarth - A World Wide Web for Robots". In: *IEEE Robotics & Automation Magazine* (2011).

[26] L. Wood. *Service robots: The next big productivity platform*. URL: http://usblogs.pwc.com/emerging-technology/service-robots-the-next-big-productivity-platform/.