POLITECNICO DI TORINO

Master's Degree in Mechatronic Engineering

Master's Degree Thesis

Indoor Navigation with Vocal Assistant

Alexa vs low-power vocal assistant at the edge

Supervisors

Prof. Marcello CHIABERGE

Francesco SALVETTI

Vittorio MAZZIA

Candidate

Giulia BERTEA

October 2021

Abstract

Among the numerous human-machine interaction methods, vocal communication has become very popular in the latest years. When initially brought onto the market, vocal assistants were strictly integrated on portable devices; nevertheless, nowadays it is becoming clear that they can be a useful feature for service robotics. In particular, driving a robot vocally constitutes a more inclusive mean of communication, which guarantees a faster and more straightforward way of asserting a command. Indeed, this technology is beneficial since it allows to tackle the needs of some social groups such as the elderly, visually-impaired or physically-limited people.

The main purpose of this thesis work is to analyze and compare two different approaches to vocal navigation, while developing and deploying both on a robotic platform for domestic environments. The first approach exploits Amazon Alexa and the AWS cloud system, to which it needs to connect. This aspect represents the greatest drawback of this approach, since it brings out many issues related to privacy and security; moreover, it requires constant internet service availability. A valuable alternative can be a low-power vocal assistant at the edge, which is therefore locally integrated on the robotic platform, that has been implemented by a team of researchers at PIC4SeR (PoliTo Interdepartmental Centre for Service Robotics). This vocal assistant is a compound of different machine learning models for speech recognition and processing.

An algorithm for the navigation of the robotic platform is developed and integrated with both vocal assistants. The main functions implemented allow the robot to follow basic navigation instructions and steer towards predefined sets of coordinates, which identify rooms and goals in a hypothetical map.

Furthermore, an analysis of the meaning extraction methods exploited by both approaches is presented. Regarding the low-power vocal assistant at the edge, a more powerful and precise module for the action classification, based on natural language processing algorithms, is proposed and integrated into the application. The described module exploits advanced machine learning techniques, such as transformers and Deep Attention Neural Networks, for encoding and classifying sentences into predefined categories of instructions.

Finally, after extensively simulating in a virtual environment a service robot guided by the two vocal assistants, some real-world tests are run with the goal of highlighting the limitations and advantages of both approaches. The results obtained open up to various future implementations and show how service robotics can highly benefit from vocal assistants at the edge, especially in indoor environments for assisting elderly, visually-impaired or physically-limited people.

Table of Contents

List of Figures

List of Tables

Part I

Introduction

Chapter 1

Thesis Objective

Even though, nowadays, industrial robotics is still dominant in the robotic research field, the trend of service robotics is rapidly growing. The interest towards autonomous mobile robots assisting human beings in their daily life is permeating every aspect of the Digital Age. More than ever, leaving to robots repetitive, monotonous or dangerous tasks, seems like a good idea for improving the lives of many. PIC4SeR, the Interdepartmental Centre for Service Robotics at Politecnico di Torino, where this thesis work has been developed, focuses particularly on service robots, trying to find new ways our society can benefit from them.

The main purpose of the thesis is the implementation of a mobile robot capable of navigating an indoor environment, following vocal commands. The approaches analyzed are two, therefore the differences between them are highlighted, as well as the advantages and disadvantages of both methods. In the second approach, the application developed is part of a broader project, carried on by researchers at PIC4SeR, which concerns the development of a low-power vocal assistant at the edge. However, the work discussed can be useful for many different applications, especially those targeted at helping elder, visually impaired or physically limited people whose autonomy in domestic environments can be improved, as well as their overall well being. Indeed, this thesis shows that, for some applications, hearing characteristics of robots are just as important as vision ones.

In conclusion, the main topics that can be found in this thesis work are cloud robotics and connected devices, regarding the first approach, while for the implementation of a local vocal assistant, in the second approach, the focus is on the speech recognition algorithms and the natural language processing field, with a particular attention to the feature extraction from commands asserted to the robot.

1.1 Thesis Outline

In this section, the author aims at giving a brief overview of the structure of the thesis and the contents of each chapter. In particular, there are four main parts.

The first part introduces the reader to the objectives, the motivations, the tools and the methodologies exploited to carry on this work. The chapters in this part are organized as follows:

- chapter 1 presents the objective and the outline of the thesis;
- chapter 2 is devoted to better explaining the goals of the thesis, explaining the two approaches used to carry on the research, and the motivations of the project to be developed;
- the state of art of speech recognition can be then found in chapter 3, with the goal of providing the reader with a small theoretical background that allows to better understand the topics of the work;
- finally, in chapter 4 the software and hardware tools exploited for the thesis work are presented.

The second and third part expand, separately, the two approaches to vocal navigation, trying to highlight the advantages and disadvantages of the two methods, and explaining the reasons why it is sometimes necessary to aim at developing a local assistant, even though more powerful ones are already available on the market.

In particular, the outline of the second part, focusing on an Alexa-driven Approach, is the following:

- chapter 5 displays the state-of-art of the two main topics of this part: vocal assistants and cloud robotics;
- in chapter 6 the author illustrates the working principle of the application, how it was developed, how the code of the Alexa skill and the ROS nodes work and how they were linked together;
- in chapter 7 the results of the simulation and testing, both on Gazebo and on the real hardware are shown and commented, as well as the limits of the approach.

The third part of the thesis deals with the second approach, based on machine learning techniques. The organization of this last section is the following:

- chapter 8 offers an overview on machine learning applied to speech recognition. Audio processing, speech-to-text applications and natural language processing are briefly presented and contextualized, showing the applications in which they are mostly deployed;
- in chapter 9 the focus is firstly on how the application developed at PIC4SeR is implemented and then on how it can benefit from an improved action interpreter. In particular, the author propose a technique based on a model offered by Google and the motivations behind this choice;
- in chapter 10, similarly as done for the first approach, the results of both simulation and testing are reported and analyzed.

The last part, made up of chapter 11 only, aims at drawing the conclusions for this thesis work, summarizing the advantages and disadvantages of the presented techniques. Moreover, possible applications and future work are also presented.

The relevant code produced by the author for this thesis project can be found in Appendix A.

Chapter 2

Vocal Navigation

In this chapter the author analyzes the problem of service robots navigation in an indoor environment, presenting the main features of service robots, the reasons that bring at implementing a vocal assistant for their navigation and the two different approaches inspected in this work.

2.1 Indoor Mobile Robots

In modern times, thanks to industrial applications and the necessity of automating monotonous and precise tasks to speed up and improve the production process, fixed robots became the most popular and the most common ones. However, in the last decades, interest towards mobile robots has started growing, together with the ambition, typically human, of creating intelligent human-like machines able to do a lot of fascinating things and make our lives easier [1].

As the name suggests, mobile robots are characterized by the ability of moving in an environment without constraints on the position and the orientation. Therefore, they are classified based on their main feature: a mobile base equipped with a locomotion system that can be either wheeled or legged [2].

In particular, the robotic platform exploited for testing the code produced by this work is called TurtleBot3 and it can be classified as a *differential drive* vehicle. Indeed, it is equipped with two parallel actuated wheels, whose independent velocities determine the motion of the robot. In particular, the motion takes place about a rotation center, defined as Instantaneous Center for Curvature (ICC), located at any point on the wheel axis and determined, in this specific case, by the ratio between the velocities of the two wheels.

Nowadays, mobile robots are mostly used as service robots, defined by the International Organization for Standardization as *"robots performing useful tasks for humans or equipment excluding industrial automation applications"* [4]. In the

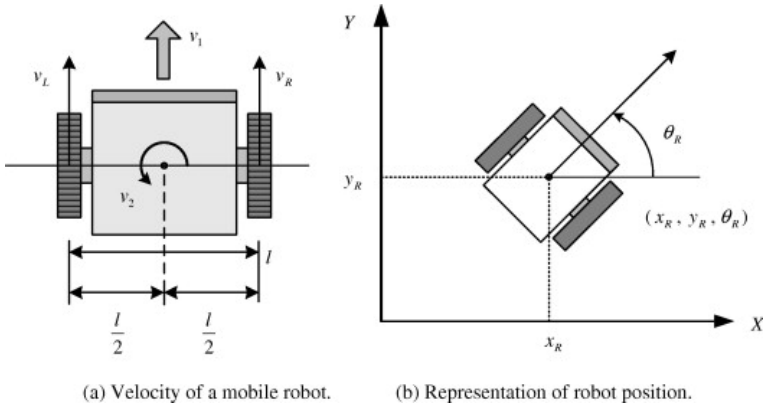


Figure 2.1: Differential Drive Vehicle [3].

last decades, indeed, the concept of robot as a purely industrial entity has been replaced by that of a machine able to help us in the daily tasks. Applications of robots in indoor environments, such as our homes, hospitals, schools, museums and so on are becoming more and more frequent and as people are starting to accept robots cooperation, experts are focusing on how to better exploit the endless resources that these technologies can offer.

2.2 Robotic Vision vs Robotic Hearing

As AVG technologies are becoming the most discussed field of application of robotics and automation, also in the literature concerning indoor mobile robots, it is very common to come across researches and studies regarding the improvement of indoor autonomous navigation systems. Moreover, many surveys can be found regarding object and obstacle detection. Hence, we can deduce that, at present time, the core of the researches in the mobile robotic field is identified by the mobile robot vision.

However, in many applications, the complete autonomy of machines in their movements can be somewhat of an holdback and the importance of being able to guide the robot according to our specific needs has been often overlooked. In particular, a mobile robot might be required to respond to our real-time demands without having the freedom to choose how to operate and move around the environment. There are many different ways in which we can control a robot and though a tactile command is usually thought of, it is often useful to take into account voice control.

Speech is the most frequently used mean of communication among human beings, therefore it should not come as a surprise that the most practical way for people to interact with intelligent agents is indeed exchanging vocal information. Besides being the fastest way of giving a command from the user's point of view, it also

provides a much larger range of people with the ability to operate a mobile robot without requiring any specific knowledge or ability besides one's own voice.

Vocal commands can be very helpful in risky situations in which men can't operate manually the robot, hence in many papers hints to spaceships appear as an example, next to the suggestion of using these applications in mines or similar environments. Moreover, indoor mobile robots that do not need to be manually controlled allow elderly, visually-impaired or physically-limited people to easily demand their needs and be helped and therefore lead a more autonomous life.

Even though it is clear that exploiting vocal commands could lead to impressive results in many applications, surprisingly this type of command assertion is not often taken into account. Indeed, many progresses have been made in the field of machine learning regarding speech recognition, but they are hardly ever applied to mobile robots.

A mobile robot able to both see, listen and react to the human with whom it is interacting constitutes the perfect combination to respond to a wide range of tasks that it could be asked to perform in an indoor environment. Hence, it would be necessary to proceed to improve both sight and hearing abilities of the artificial machines in parallel. Since many papers already deeply explored the first one, the focus of this work is precisely to show how it is possible to exploit vocal commands to navigate a mobile robot in an indoor environment and expose the benefits we could derive from it.

Obviously, it is of primary importance to improve the speech recognition readiness and accuracy, particularly exploiting cloud services and IoT services. Moreover, another focus of research in recent years is on the development of lightweight applications and the use low-power hardware platforms. Finally, the author underlines the importance of integrating in these kind of applications with a vision system, not only to achieve a more complete robotic platform, but also for safety purposes.

2.3 Two Approaches to Vocal Navigation

In this work two different approaches to the problem of navigating a mobile robot giving it vocal commands are considered:

- Alexa-driven Approach;
- low-power vocal assistant at the edge.

The first one focuses more on the navigation algorithm, exploiting an existing and ready-to-use vocal assistant, whereas the second approach aims at developing a specific low-power assistant locally deployed on the hardware for the same tasks.

In particular, the first approach investigates the link between robotics and cloud systems, analyzing the available tools for facilitating the integration of the vocal

assistant with the hardware, but also highlighting the limitations and risks deriving from the approach.

The second approach concerns many aspects of speech recognition: key-word detection, speech-to-text translation, action interpretation, voice generation and so on. The author of this work focuses specifically on the way commands are interpreted by the vocal assistant and associated to a specific action, analyzing different meaning extraction and text classification methods.

After simulating and deploying both applications, the author also analyzes the pros and cons of both approaches in order to try to explain their validity according to the needs of the user.

The main advantages of exploiting Alexa, for asserting vocal commands, are given by the great power and accuracy of this vocal assistant. Indeed, it is created by highly skilled developers using vast datasets and a great amount of resources. On the other hand, the limitation of Alexa is the stringent requirement of connecting to the internet and specifically to the cloud system.

In many occasions, the limitation described imposes the need of exploiting a local vocal assistant. The advantages of having an integrated assistant are often more than the limitations given by the little quantity of dataset and resources available in this case. Especially for applications needing a limited amount of commands, using low-power assistant directly available on the robotic platform allows to have a faster, safer and always-available robot driven by voice.

Chapter 3

State of Art in Speech Recognition

The goal of this chapter is summarizing the state of the art of technologies based on speech recognition and Natural Language Processing (NLP), such as voice-based machines, intelligent agents, chatbots. The applications in which these technologies have been already exploited or could help greatly in the future will be analyzed as well.

3.1 Voice-based Machines

In recent years the progresses made in the disciplines of signal processing and machine learning combined allowed to explore the possibility of communicating with machines. Until then the concept of robot as a humanoid had only appeared in sci-fi movies, while in the real world machines were only thought of as manually operated tools making some tasks easier for us. Adding to machines the abilities to listen and to respond finally broke the barrier between fiction and reality. Hence, probably influenced by scientific novels, we started feeling threatened by what robots could accomplish and therefore we started calling them assistants, clearly stating that they only existed to serve us.

Modern intelligent virtual assistants are equipped with voice users interfaces that allow a more user-friendly human-machine interaction. In particular, the first chatbot capable of being vocally interrogated was Siri, installed on the iPhone 4S in 2011. Siri was followed after few years by Alexa, Cortana, Google Assistant and so on and therefore the interest in these newly developed and succeeding technologies started to expand.

While at first, vocal assistants were mainly available on mobile phones or laptops, eventually they were able to help greatly with a larger variety of tasks.

They were found to be particularly useful especially in indoor environments such as homes, healthcare centers, schools, whereas some categories of people could benefit significantly from this interaction mode.

3.2 Speech Recognition and Processing

Speech is the main mean of communication among human beings, therefore ever since machines were first invented men have always been striving for interfacing with them in the same way they interacted with other people. For this reason the literature on speech recognition is very extended, although it wasn't until recent years that major steps were taken and human beings actually started giving vocal commands and getting conversation-like answers from machines.

The rapid growth and improvement of vocal recognition applications goes side by side with the rise of a new branch in the machine learning world: deep learning. Approximately after 2006 many deep learning algorithms were refined and allowed to solve the issue of extracting precise features from artificial neural networks (ANN) [5]. Another field of research that had great improvements with the breakthrough of DNN is that of natural language processing, closely connected to speech processing, because it allows the machines to understand the requests of the user, by creating machine-readable representations of the language.

Indeed, one of the most challenging aspects of this field is trying to put together a concept as complex and polyhedric as the human language. While a simple picture can be modeled in an easier way by means of shapes and colors, there are many more aspects that need to be taken into account when dealing with human languages. First of all, language is not unique, there are more than 7000 languages known in the world; moreover languages are not static but they dynamically change from a domain to another and they develop in time as well. It is exactly for these reasons that if on one side language is very complex and hard to model, on the other side it allows us to access a great variety of information.

Hence, the union between speech recognition and language processing can be exploited in many different ways: to extract information on the content, the language, the accent but also features of the speaker such as the gender, the age, the identity, the emotional status, the health status, but also to translate, summarize or extract key-words from an audio file. These considerations highlight how complex the field of voice detection is and they suggest that many different methods must be investigated to understand which one better suits the application in order to obtain the needed result.

At first, the most popular method for dealing with speech recognition, before the breakthrough of deep learning, was the representation of signals through hidden Markov models (HMM). However the main limit of this simple and practical

model was constituted by statistical inefficiency in many applications concerning non-linearity situations.

It is important to highlight how these techniques were not abandoned after the power of neural networks was discovered. Actually, especially at the beginning, many obstacles were found in using neural networks, therefore, they were often combined with other models such as HMM, offering them a way to pre-elaborate the input data.

Deep learning algorithms differed from other algorithms because they better adapted to the goal of modeling the complexity of language. Even though this field of artificial intelligence always existed, it is important to detach it from the idea of programming a machine and embrace the idea of teaching it how to learn a pattern from the input data instead.

3.3 Applications of Speech Recognition

As illustrated before, automatic speech recognition (ASR) allows to extract countless features and so it has very diverse applications, such as automatic writing, automatic translation, chatbots for improving costumer experience, smart homes, speaker identification, different applications in the detection of diseases or similarly of the emotional status of the speaker.

Moreover, even more applications of speech recognition can be thought of if the ability of the intelligent agent to hear and understand a vocal instruction or a question is put together with the ability of not only respond but also act upon the user's request.

This area of research has not been widely explored yet; in the last decades researchers undeniably focused their attention on achieving autonomy of mobile robots. Even if, on the other hand, another key point of scientific investigations has been the development of intelligent assistants, the two fields hardly ever collapsed together to generate a truly complete machine from the human interaction point of view.

Mobile robots that can be controlled by human voice, though, have a great area of usage, such as in risky environments or in improving the life of old, physically limited or visually impaired people. They have also been found to be very effective to help people with dementia or other generative diseases in remembering things and also in feeling less lonely, with many beneficial results deriving from the use of ASR.

Chapter 4

Software and Hardware Tools

In this chapter the author wants to focus on the description of the software and the hardware platforms used to simulate the speech recognition and navigation applications presented in the next chapters.

First of all, the main set-up used for coding and simulating was a machine with an open-source Linux-based OS, in particular Ubuntu Bionic Beaver 18.04. For the prototyping and testing of the code, the hardware used is described in the following paragraphs, as well as the software used for the application.

A new robotic platform, named *Jetzero*, is obtained by the compound of a TurtleBot, a Jetson Nano board and Zero vocal assistant.

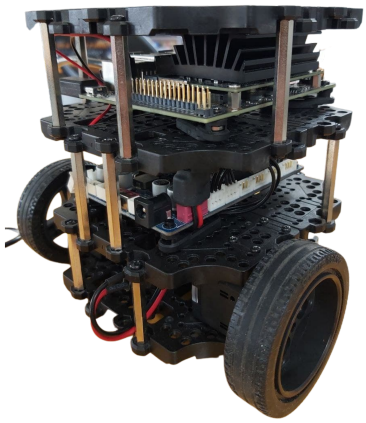


Figure 4.1: Hardware platform used for testing made up of TurtleBot3 Burger and NVIDIA Jetson Nano.

4.1 Robot Operating System

ROS is an open-source meta-operating system for robotic platforms of many kinds. A peculiar characteristic of this meta-OS is that it is real-time and it is based on a distributed peer-to-peer architecture of nodes, for this reason it can be easily represented by means of a graph of nodes and arcs.

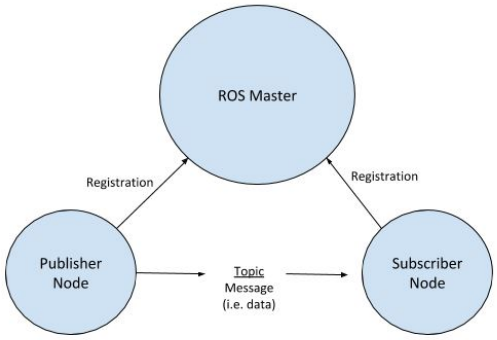


Figure 4.2: Communication Scheme of ROS Nodes and Topics.

Each node is a running process that needs to be registered to an always-running Master node. The aim of the Master is only that of monitoring the information exchange between the other nodes, which, though, communicate directly with each other, without interfering.

The communication between nodes takes place by means of topics or services. In this thesis only topics where exploited and are briefly illustrated. The main idea is that the node sending a message, called *Publisher*, needs to set a "channel" called, indeed, *topic*. Afterwards, the Publisher node can start publishing messages of different types on it. On the other hand the node that should receive the messages, subscribes to the topic in order to be able to read the messages, for this reason it is called *Subscriber*.

The advantage of ROS with respect to other robotic frameworks is that it provides many libraries and tools for programming a robot, offering a huge community useful for support and for the reusability of the applications.

4.2 TurtleBot3

TurtleBot3 is the last version of a ROS-based mobile robot used mainly for education and research purposes, whose advantages are that it is small, low-cost and programmable [6]. There are different kinds of *Turtlebots*, with diverse shapes and

characteristics. For this thesis work a *burger* type was used, since due to its small size it is the most similar one to a common home service robot.

Figure 4.3: TurtleBot3 Burger.

The main characteristics of this simple, but very useful for prototyping, robotic platform are:

- dynamixel actuators;
- an embedded controller OpenCR;
- a Single Board Computer (SBC).

Moreover, TurtleBot3 offers the possibility to easily test the application in Gazebo thanks to the availability of a package containing many testing environments and tools for testing the code even without deploying it on the real hardware.

4.3 NVIDIA Jetson Nano

Jetson Nano Developer Kit is a SBC by NVIDIA, only consuming 5 W to run, which contains in little space everything that it needs to efficiently run artificial intelligence applications.

Thanks to its small size and high computational power, it is suitable for service robotic applications which need to efficiently perform neural network computation while exploiting an hardware that can be easily placed on board of a robotic platform.

GPU	128-core Maxwell
CPU	Quad-core ARM A57 @ 1.43 GHz
Memory	4 GB 64-bit LPDDR4 25.6 GB/s
Storage	microSD (not included)
Video Encode	4K @ 30 | 4x 1080p @ 30 | 9x 720p @ 30 (H.264/H.265)
Video Decode	4K @ 60 | 2x 4K @ 30 | 8x 1080p @ 30 | 18x 720p @ 30 (H.264/H.265)
Camera	2x MIPI CSI-2 DPHY lanes
Connectivity	Gigabit Ethernet, M.2 Key E
Display	HDMI and display port
USB	4x USB 3.0, USB 2.0 Micro-B
Others	GPIO, I^2C, I^2S, SPI, UART
Mechanical	69 mm x 45 mm, 260-pin edge connector

Figure 4.4: Jetson Nano Developer Kit Specifications [7].

4.4 Gazebo

As mentioned, Gazebo is another important open-source tool, commonly used in combination with ROS, that offers a virtual environment for the simulation of the behaviour of the robot and the functionality of the code. In particular it allows to simply set up a customized simulation indoor environment and it also offers a wide variety of models of robotic platforms, such as the model of TurtleBot3, used for this work.

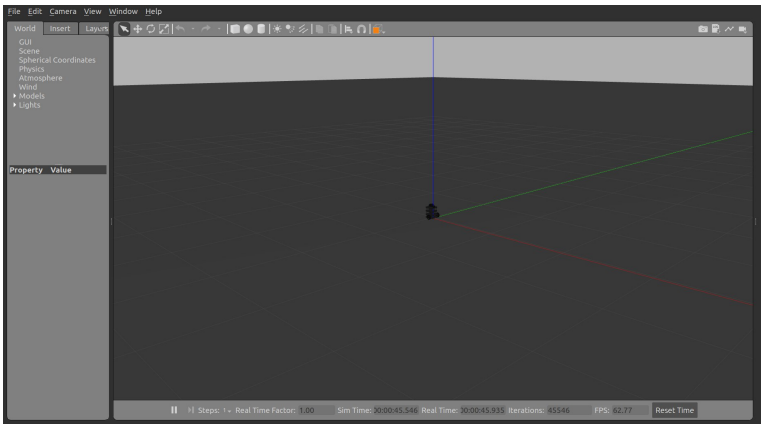


Figure 4.5: Gazebo Simulation Environment with TurtleBot3.

4.5 Tensorflow

One of the most popular software tools exploited by developers for machine learning purposes is Tensorflow, an open-source platform released by Google in 2015. In this work, already trained model are mostly used, however tensorflow is necessary for running all of them and especially the Universal Sentence Encoder exploited in the last part of this work.

Part II

Alexa-driven Approach

Chapter 5

Vocal Assistants and Service Robotics

The history and the state of the art of vocal assistants is presented next, with a specific focus on Amazon Alexa and how the cloud systems AWS supports this vocal assistant for robotic and IoT applications.

5.1 History of Vocal Assistants

Many researches show how smart speakers are becoming common devices that can be easily found in the majority of middle-class households; in particular some surveys estimated that by 2022 half of the families in the United States will at least own either a Google Home or an Amazon Echo device [8]. Moreover, nowadays vocal assistants are also integrated in smartphones and computers of any kind and therefore a great part of the global population is able to daily interact with artificial agents by speaking.

Men desired to vocally communicate with computers ever since these were invented; many older books and movies suggested that we could benefit greatly from vocal human-machine interaction on many levels. The first machine able to listen to human voice was invented in 1952 and it was called *Audrey*, a single-speaker speech recognition system that could achieve an impressive accuracy of 90%. The substantial limitation of this technology was the fact that it could only understand the voices of its creators, nevertheless at the time it was still a great invention that amazed many [9].

In the 90s many projects were carried on regarding rooms with vocally guided lights, speakers, blinds and so on. The most famous projects in the universities of the United States in those years were Intelligent Room by MIT, ComHOME by the Interactive Institute, Aware Home by Georgia Tech [8]. However these projects had

no long-term application, since they were based on a large amount of computers and the performances were not very satisfactory.

It was only in recent years, with the improvement of the accuracy of speech recognition and the general advancement of the hardware components and their lower cost, that vocal digital assistant could be deployed in a way that could be accessible to the population. The history of voice-controlled assistants actually began in 2010, when Apple introduced Siri as a standalone app at first and then integrated in iOS shortly after.

The revolution of Siri, compared to earlier technologies that could respond to vocal commands, was the ability to understand a larger variety of commands and to perform all kinds of tasks. The user was finally given the freedom to ask for help as it was speaking to another human being, without worrying about whether or not the machine could understand the given commands and without needing a limited list of tasks that it could operate. This achievement was possible thanks to artificial intelligence as well as the continuous connection of the digital assistant to the internet.

After Siri's appearance and success on the market, many other assistants started to be presented by Apple's competitors. Even though the purpose of these assistants is similar to Siri they are equipped with different voices, features and available tasks. In 2013 Microsoft created Cortana, then, a year later, Amazon launched Alexa together with Echo, a specific home speaker in which it was integrated, then later in 2016 Google's Assistant and Google Home were introduced on the market as well.

Even though the availability of these technologies has increased a lot in the latest years, studies show how vocal assistants are not exploited as much as they could. Since they provide simply a bridge between the user and the app, many still prefer to issue commands manually, to avoid annoying situations in which the vocal assistant does not rapidly understand and respond.

However, as speech recognition technologies are improving on a daily basis and the field of application expands, vocal assistants are becoming very useful in the daily life of young people and are destined to be exploited more widely, even for the elderly community.

For example they can be a great help both for people that have an hard time issuing commands manually and for people who need constant reminders or even just some company at home, such as elder people. Finally, voice assistants could be exploited widely by businesses to answer common questions of the costumers or even for letting the costumer buy products directly with a vocal request, as for example Starbucks does with its famous coffee drinks [10].

5.1.1 Amazon Alexa

Amazon Alexa was released in November 2014 together with Amazon Echo device on which it was deployed and a smartphone app for managing settings. Nowadays, it is estimated to be supported by a hundred thousand devices worldwide, being the most deployed vocal assistant in the world.

Besides being exploited for basic tasks such as setting timers, putting music on and sending text messages, Amazon Alexa has been the first vocal assistant widely exploited for home automation as well. Compatible devices for smart homes are smart lights, plugs, locks, security cameras, wi-fi routers and so on and allow the user to vocally control every functionality of the house.

The advantage of Amazon vocal assistant is the great availability of skills offered. Even though it was launched after many other vocal assistants, Amazon was the first one to offer to the user the possibility to develop custom third-party skills for Alexa, as well as simple routines and personalized tasks.

A study published in 2019 (Figure 5.1) analyzed the growth of available skills of Alexa over three years, from 2016 to 2019. The trend grew very rapidly, as the number of skills went from about 130 to over 100 thousand over that relatively short time, and it increases daily thanks to users around the world. The same study also shows that the countries from which most of the functions come from are the United States, the United Kingdom and India [11].

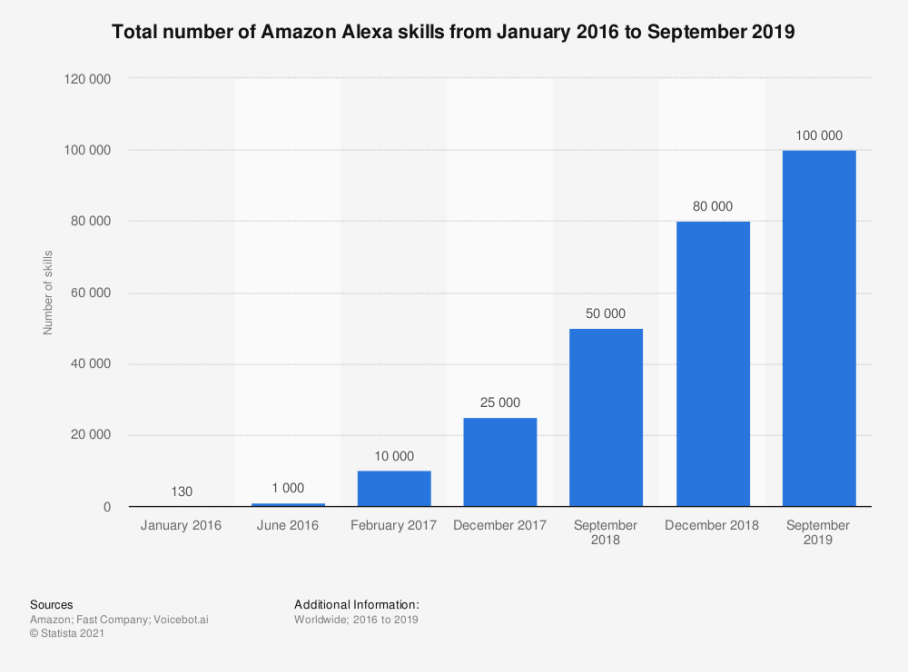


Figure 5.1: Alexa Skills Trend from 2016 to 2019 [11].

5.2 Internet of Robotic Things

In 1999, the English engineer Kevin Ashton was the first one to coin the expression *Internet of Things* (IoT). Nowadays, people are used to hearing the acronym IoT, even though the majority probably could hardly guess what those three letters stand for. The Internet of Things can be described as the set of all the physical things that are connected to the internet and communicate and exchange information thanks to this connection.

The International Telecommunication Union (ITU) defined the Internet of Things as a *"global infrastructure for the information society, enabling advanced services by interconnecting (physical and virtual) things based on existing and evolving interoperable information and communication technologies"* [12].

It is necessary to clarify that these *things* are more than smartphones and computers, the IoT actually comprehends a fast increasing number of *smart objects* that surround us in our homes, offices and public places with the goal of automating and improving the performances and the user experience of those objects. Recently, a new concept related to the IoT world was introduced: the Internet of Robotic Things (IoRT).

A common definition has not been given yet, however IoRT can be easily characterized as the union between internet connection, cloud computing and robotic entities. The role of the robot is clearly that of a smart object that can benefit from the connection to the internet in many ways, being able to establish a communication with other smart objects, to take advantage of a greater quantity of data and to be controlled even remotely.

The necessity of connecting robots to the internet has arisen from the limitations presented by networked robots. The physical constraints, such as low speed, low memory, latency and lack of intelligence was mainly related to the hardware. One of the first examples of application of this new concept was the *Mars Rover* that needed to be remotely guided through the exploration of Mars [13]. Even if with time the availability of powerful hardware increased exponentially and the cost decreased significantly, it can be often advantageous to exploit *cloud robotics* for many reasons.

First of all, the breakthrough of machine learning in recent years allowed robots to exploit learning algorithms rather than traditional programming bringing the focus on the necessity of even more powerful hardware in terms of computational effort, memory and speed. Indeed, machine learning always requires faster and more efficient GPUs and a great amount of data for the training process. These factors can lead to a cost of the hardware that is often overwhelming and not worth it for many applications.

Another aspect in favor of cloud robotics is given by the reusability of the resources. Many cloud services allow to share code, applications and generic

knowledge among the community, reducing the developers' effort and speeding up the progresses in the robotic field. Whereas the need of an internet connection always available may seem mostly a limitation, it can actually help the developers and researches in many cases.

5.2.1 Architecture of IoRT

As previously described, IoRT is not a new technology, but simply a compound of many existing ones put together with the goal of exploiting cloud services and large amount of data available online.

The parts composing an Internet of Robotic Things platform, can be described as five layers put one on top of the other exchanging data and messages of different kinds and exploiting various protocols. In particular, these layers are similar to those described in the literature of internet of things and can be classified as follows.

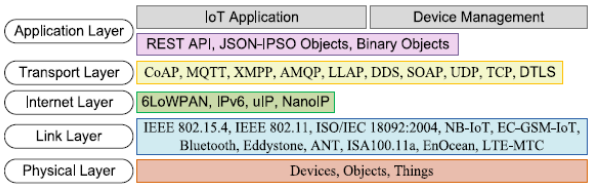


Figure 5.2: Layers and Protocols of IoRT [13].

- the hardware layer (comprising physicals components such as vehicles, robotic parts, sensors, actuators, computers, smartphones, speakers);
- the network layer (defined by the connectivity system that can be of different kind, for example cellular, wi-fi, bluetooth, NFC);
- the internet layer (it is the core of the architecture and it is made up of communication protocols such as MQTT, IPv6, CoAP, DDS);
- the infrastructure layer (made up of the robotic platform support, usually ROS, together with a cloud platform service);
- the application layer (actual application code to be deployed for the specific usage).

Chapter 6

Alexa Vocal Navigation

In this thesis work the main goal, as previously stated, is the navigation of a mobile robot through an indoor environment. However, the peculiar aspect of this project is related to the way commands are asserted: using voice and allowing an hands-free driving system.

The first approach proposed is based on an existing vocal assistant, developed by Amazon, in order to focus at first on other aspects of the project. The availability of Alexa, connected to TurtleBot3 thanks to an internet connection and the AWS cloud services, allows to promptly navigate a mobile robot following vocal commands, exploiting a fully developed intelligent agent.

6.1 Working Principle

The overall idea of this approach is to send the navigation commands to the AWS cloud system exploiting a custom Alexa skill and to then receive them on the service robot itself where ROS nodes are ran and allow to send velocity commands and move the robot. In particular, two nodes are necessary, a first one serving as a communication bridge between MQTT messages and ROS messages and a second one sending messages to move TurtleBot3.

As previously mentioned, in order to connect all these components and applications together, AWS services, such as Lambda and IoT Core, have been used and therefore the author will firstly describe how they work and how they communicate with each other.

The comprehensive scheme shown next (Figure 6.1) aims at explaining the way vocal commands, translated into text by Alexa, are sent to the mobile robot. On one side a *human speaker* addresses its vocal commands to an *Amazon Alexa device*. Therefore, an Alexa skill, whose code is stored in *AWS Lambda*, is invoked. On the other side *TurtleBot3* runs the ROS nodes and thanks to the *mqtt_bridge* it

connects through MQTT protocol to another AWS service, *AWS IoT Core*, that is easily linked to the skill using the *endpoint* and *certificates location* of the defined *thing*.

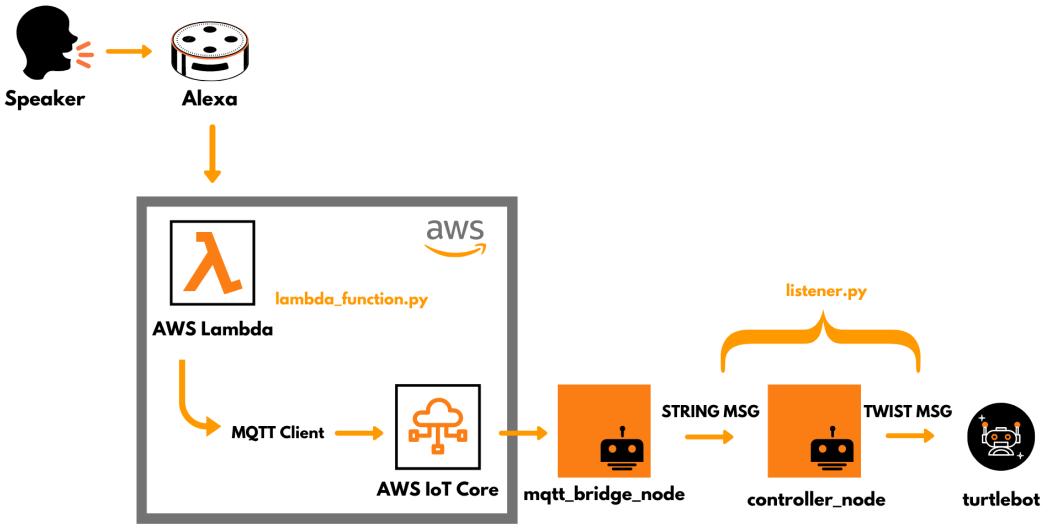


Figure 6.1: Application Working Scheme.

6.2 Alexa Skill

Since the aim is driving TurtleBot3 by exploiting an existing vocal assistant, the choice of using Amazon Alexa derives from the fact that it allows the user to easily develop personal and customized skills activated with one's voice. More specifically the Alexa Skills Kit (ASK) is a software development framework that allows to define the structure, write the code and manage the settings of deployment of a personalized skill.

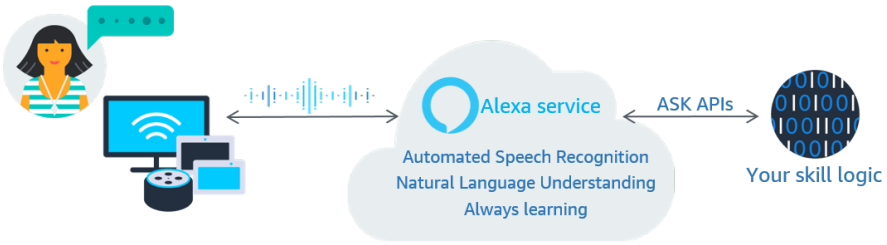


Figure 6.2: Alexa Skill Kit Framework [14].

It is possible to develop a custom skill on the *Alexa Developer Console* following a specific workflow to build the interaction model. In particular, the different phases are:

- design
- build
- test
- certify
- publish

The most important phase is the first one, in which it is necessary to define the interaction model of the skill. The skill developer is asked to decide the sentence for the *invocation*, as well as the *intents* and *slot types*. The first one establishes the key sentence to open the desired skill, while the other two allow to define the interface for the user-skill interaction, such as the invocation sentences of the possible tasks to be executed and other variables. All of these components together form the action interpreter of Alexa, which is based on complex speech recognition, as well as NLP algorithms.

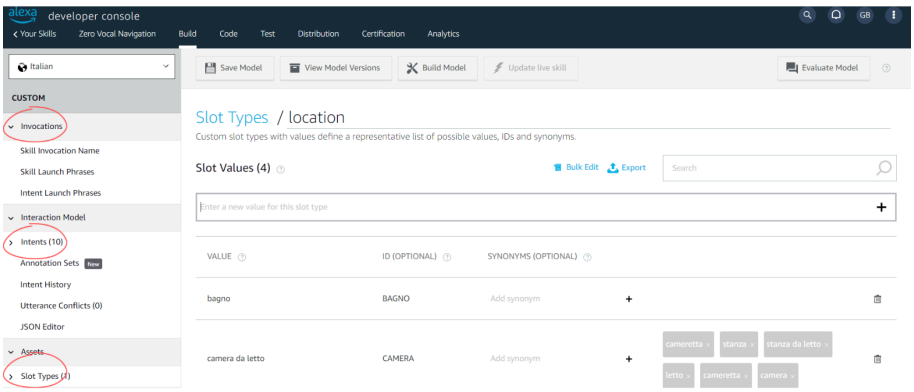


Figure 6.3: Alexa Developer Console.

In the same phase other important parameters are set, among which the *endpoint*. In this case the endpoint type is set to AWS Lambda ARN and the address is substituted by the ARN identifying the lambda function in which the skill code is stored and that will be explained later.

6.2.1 Action Interpreter of Alexa

Thanks to the tremendous amount of available resources in terms of data and computational power, Amazon Alexa is a very effective vocal assistant. A complex set of *action interpreter* algorithms allows it to understand one's request without prompting errors most of the times. Nevertheless, it is important to distinguish between the general purpose services provided by Amazon to every user and the customized skills that the user can implement by themselves, since in the last case some flexibility and efficiency of the intelligent agent is lost.

When dealing with custom skills, the interaction model that can be implemented is simple and user-friendly, based on two different types of data: intents and slot types. Basically, when creating a new skill the developer decides all of the possible actions offered by the skill, as well as all of the utterances invoking each action. Slight differences in the utterances are accepted by the vocal assistant, however it does not offer great flexibility and often calls the *Fallback Intent* whenever the sentence received is not recognized.

Moreover, slot types allow the developer to define a variable element inside an intent, however types not explicitly defined are not understood by Alexa, causing a high level of inflexibility to the application and a loss of *intelligence* of the machine. This happens because the recognition of the action relies on the developer definitions and not on the machine learning applications.

6.3 Amazon Web Services

Amazon Web Services (AWS) is a cloud platform owned by Amazon and launched around 2002. It provides many cloud computing services and APIs for a wide range of applications. It is available to anyone that registers to the website and offers many plans that allow to benefit from a great quantity of cloud resources.

In this work AWS was exploited for creating and deploying the custom Alexa Skill on the service robot. In particular, the used services are called AWS Lambda, for the storage of the code of the skill, and AWS IoT Core, that allows to connect smart things, such as the computer for simulation and the mobile robot for testing, to the skill.

6.3.1 AWS Lambda

AWS Lambda is a web service exploited in our project for storing the skill code. In particular Amazon Developer Console allows to link a custom skill to AWS through an ARN code, so that the written code can be *serverless*. More in general, AWS Lambda is a computing service that allows to run some functions, called lambda functions, automatically managing the resources required by the code.

Basically a zip folder, containing the Python script that defines the skill behavior (*lambda_function.py*), is uploaded together with the packages needed to run the code. The code of the lambda function alone can be found in the Appendix [1].

6.3.2 AWS IoT

Another important service for cloud-robotic applications is AWS IoT, the Amazon service for Internet of Things, that indeed allows to manage all of the physical *things* that need to be connected to the internet. In particular it offers the possibility of registering the hardware object that needs to be controlled remotely and of connecting it to a customized lambda function; moreover, it allows a secure way of exchanging messages between the cloud and the robotic platform.

In details, for the simulation it is necessary to register the computer on which ROS nodes are run, while for the deployment it is substituted by the Jetson Nano board, which is directly connected to TurtleBot3, that runs the nodes locally.

While registering the needed thing, the following important parameters and files are generated and are exploited for connecting the thing to the lambda function:

- endpoint of IoT Core;
- certificates and private key of the thing.

Moreover, it is necessary to attach a policy to the created thing, which authorizes the access and therefore the communication thing-function.

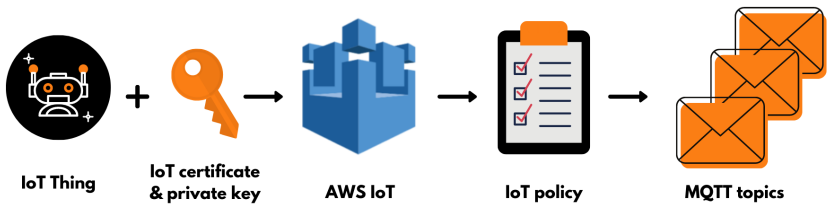


Figure 6.4: AWS IoT Core.

6.4 Communication Protocol

The communication between the thing and the lambda function is done through the *MQ Telemetry Transport* (MQTT) protocol, and specifically the x.509 certificate is used. The MQTT is a lightweight messaging protocol often used in the robotic field. Even though, the way messages are published and subscribed in the MQTT protocol is very similar to the way messages are exchanged on topics by ROS nodes, a ROS node can't simply subscribe to a MQTT message.

An existing ROS package, *mqtt_ros_aws_iot* [15], that serves as a bridge between MQTT and ROS was cloned and used to transform the MQTT messages in ROS messages.

6.4.1 MQTT

As previously stated, the MQTT protocol is a communication protocol, which was designed in 1999 by an IBM engineer. Contrary to HTTP that is based on a request-response model, MQTT exploits a simpler and lighter publish-subscribe model.

This Machine-to-Machine (M2M) protocol was first invented for transferring messages efficiently, even with very little bandwidth, to one or multiple clients. Nowadays it is widely used for IoT applications, mainly because of its characteristic of being lightweight and power-saving. Indeed, MQTT is suitable for those applications that require low impact and in which the amount of bandwidth available is very limited.

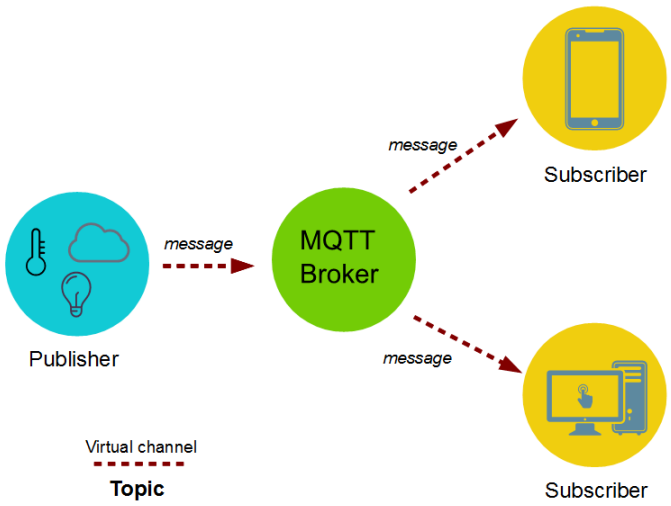


Figure 6.5: MQTT protocol [16].

Moreover, the publish-subscribe model does not require a direct connection between publisher and subscribers and guarantees that the communication is asynchronous as well. Much like radios and TVs, this protocol allows the publisher to simply "broadcast" the messages and the subscribers to "view" the topics when needed. For this reason, a central *broker* is required for the functioning of the MQTT protocol, as shown in picture 6.5, for filtering messages and distribute them to the subscribers.

In particular, various MQTT protocols are supported by AWS and therefore it is possible to exploit them for transferring the information received through the Alexa Skill to the connected smart thing, such as the service robot (subscriber). Finally, as mentioned, the MQTT bridge node translates the messages on the topics of the MQTT protocol in messages that can be sent over ROS topics, such as string messages.

6.5 ROS Nodes and Topics

A ROS node is used to actually drive the robot. It is called controller node and it publishes the topic *cmd_vel*, on which *Twist* messages are sent. These messages describe the velocity of the robot in all the directions and are set differently according to the received message.

In practice, the node periodically subscribes to the topic published by the MQTT bridge node *aws/iot/available*, describing the text received by Alexa and transformed into string type data. The callback function of the subscriber then calls other functions according to the information contained in the received string and the twist messages are generated and published on the *cmd_vel* topic. Finally, the *odom* topic is exploited as well by the controller to receive continuously the updated position of the robot and navigating towards a goal.

In figure 6.6, it is reported the ROS computation graph, called rqt graph. It allows to understand how the communication between the nodes happens. In particular, it shows that the velocity topic is published by the controller node and subscribed by the simulation environment node (*gazebo*) in which the the robot is simulated. However, at the same time this node acts as a publisher, allowing to send back to the controller node messages carrying the position of the simulation robot, in a retroactive way.

Figure 6.6: Rqt Graph.

6.5.1 Controller node

The core of the application, whose complete code can be found in the Appendix [2], is constituted by the *controller node*. It represents the control unit of the robot, which receives the text data containing the commands and translating it into velocities sent to the motors and driving the robot.

Besides the basic commands that allow to vocally direct TurtleBot3 to a needed point in space, with a step-by-step commands procedure, navigation towards a set of given coordinates has also been implemented.

The implemented ones are just examples of commands that can be useful for any general purpose service robot navigating exploiting voice control. Many others can be added, simply adjusting the code according to the needs of the application.

For example, the command *follow me* or *save location* can be handy for these kind of robotic platform, however they require vision and mapping tools as well. In the table below the author reported the illustrative commands implemented for testing the application on TurtleBot.

BASIC COMMANDS	LOCATION COMMANDS
avanti	cucina
indietro	salotto
sinistra	bagno
destra	camera da letto
fermati	base

Table 6.1: Commands for TurtleBot3 Vocal Navigation.

The first goal could be integrated on any service robot that could receive velocity commands on all three axis, even if not endowed with vision capabilities. In the case of straight line navigation, only the linear velocity along the x axis is asserted, while for rotating left and right the rotational velocity along the z axis is changed. Visual aid to the robots would need to be added mainly for safety reasons, but are not strictly necessary for the implementation of these basic commands.

On the other hand though, navigation to a specific goal defined in space by a set of coordinates, would need to be integrated with other vision applications to fully function: mapping, path planning and obstacle detection. In this thesis work the author's focus is the navigation through vocal commands, therefore it is implied the assumption of having a map available with target spots that can be reached by the robot without bumping into any kind of obstacle.

Therefore for simulating the navigation towards a specific point in space, such as a room in the house (e.g. "go to the kitchen"), it is assumed that the room is identified by a set of coordinates in a given map. In particular, a function called *mapping* links the name of the location to a random point in space. Obviously this

method can be exploited only for simulation purposes and it is an end to itself: more sophisticated mapping and navigation algorithms, based on vision, should substitute this function to have a complete robotic platform.

In order to reach these target points in space, an important aspect to take into consideration is that the velocity of the robot needs to be set according to the position of it with respect to the goal. To acquire these information on the robot position, the author exploited a topic available for TurtleBot3: the odometry topic. There are more effective and accurate ways to get its position, however, as underlined before, these aspects are beyond the scope of this thesis.

Chapter 7

Alexa Approach Simulation and Testing

This last chapter of part II aims at presenting the results obtained from the simulation and testing of the Alexa-based application. Moreover, some considerations about the advantages and, more importantly, the limitations of this approach are reported.

7.1 Gazebo Simulation

In order to simulate and check the correct functioning of TurtleBot driven asserting commands to Alexa, it is necessary to install on the machine the package containing the simulation environments of TurtleBot3. Furthermore, since the focus is on the right command-action association, a simple *empty world environment* has been used. Therefore, for testing whether the established room was reached, the coordinates chosen were integer numbers, simply marked with a "light-spot" available in the Gazebo environment.

For running the simulation two ROS nodes need to be launched, together with the previously described simulation environment. The commands to run the communication node and the controller node are the following:

- *roslaunch mqtt_ros_aws_iot connect.launch*
- *roslaunch vocalbot vocalbot_launch.launch*

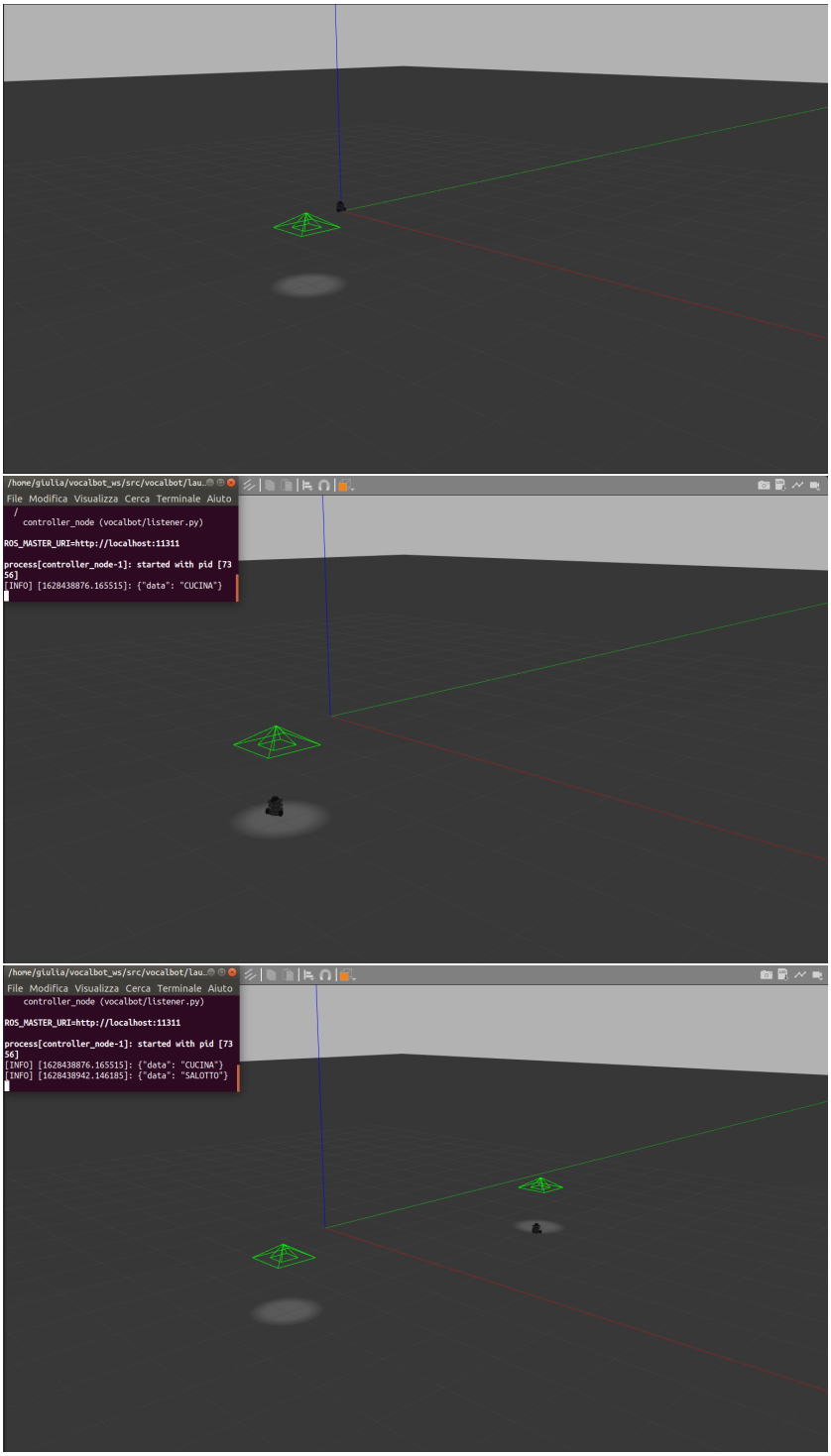


Figure 7.1: TurtleBot3 in the starting point (0, 0), in the kitchen (3, -3) and in the living room (3, 3) respectively.

In figure 7.1 an example of simulation is reported, showing the service robot going from its starting point to the kitchen and then directly to another location, the living room. As can be seen, the set of coordinates is not perfectly reached by TurtleBot, since an accuracy range must be defined before running the code, in order not to have the robot continuously changing its orientation, slowing down too much. However, the set of coordinates should identify a room, which is greatly bigger than the accuracy range, and therefore the little error can be neglected for the purpose of the application.

7.2 Real-time Testing

After checking the correctness of both the MQTT communication and the controller node behavior, the next step is the testing on the real hardware. TurtleBot3 was equipped with the NVIDIA Jetson Nano board, assembling them together in a unique robotic platform.

Since Jetson Nano runs Ubuntu, the author was able to transfer the whole *vocalbot_ws* workspace to it and then run another simulation. Indeed, the nodes could be launched on the NVIDIA board, while launching TurtleBot in Gazebo at the same time, simulating once again on the real hardware. Although some delays were detected, the simulation worked well and therefore the testing phase could be done next.

For testing the overall behavior of the service robot guided by Alexa, the same two nodes need to be launched again (exploiting an SSH protocol in order to connect to the Jetson Nano board):

- *roslaunch mqtt_ros_aws_iot connect.launch*
- *roslaunch vocalbot vocalbot_launch.launch*

However, instead of launching the Gazebo environment, in order to navigate the real robot, it is necessary to *bring-up* TurtleBot through the command:

- *roslaunch turtlebot3_bringup turtlebot3_robot.launch*

The testing was carried-out marking on the ground the coordinates associated to a specific room, assuming once again that no obstacles are present and so no path-planning is needed. The task of reaching the goal after receiving the instruction from Alexa was completed for any coordinate, with an accuracy slightly greater than the one used for the simulation. This small error is due to issues related to odometric accuracy, that can be improved with techniques largely analyzed in the state of art of autonomous navigation.

7.3 Limitations

The focus of this section is the analysis of the advantages and disadvantages of the first approach to vocal navigation presented. Although, this solution may seem the best fit for the purpose, due to its simplicity and accuracy, it is not suitable for many applications and often needs to be substituted by or sided with a local vocal assistant. The reasons behind the choice of implementing a low-power speech-to-text vocal assistant, even when disposing of powerful assistants, such as Alexa, are presented next.

Amazon Alexa offers an easy way to allow anyone to make use of a vocal assistant through a simple developer platform. Moreover, since it was developed by teams of engineers and AI experts with powerful hardware and software resources available and trained on a great vastity of dataset, Alexa offers great accuracy and high performances that are difficult to achieve with a self-made vocal assistant.

On the other hand, it is important to notice that Alexa is a general purpose assistant. It is built to adapt well to any application and context. However, since it was not built for a specific application, it could not be perfectly suited for any language domain. In particular, a specific purpose vocal assistant, could be tuned through transfer learning techniques on specific set of data, achieving better results in the NLP area.

Furthermore, the biggest disadvantage of exploiting Alexa is the need of always connecting to the internet. This may not seem a big issue, however it could limit greatly the usage of the service robot equipped with the vocal assistant.

Many environments in which these kind of robotic platforms may be exploited could indeed not have an internet connection available. Just think of an elder's home that might not be interested in having an internet connection or, for examples, hospitals, healthcare centers, schools and in general public places, where an internet connection is usually also protected by many protocols that do not allow an easy and consistent connection. Moreover, assuming the indoor environment at issue has internet connection available all the time, even a temporary lack of service could prevent anyone from guiding the robot vocally, sometimes even putting in danger people relying on its usage.

In order to bring a tangible example, when testing TurtleBot with Alexa, the author encountered issues with the internet connection of Politecnico di Torino. Indeed, the MQTT connection was blocked and therefore, the robotic platform could not receive any command. However, when switching to a different connection, the application worked.

Finally, another important issue related to the need of connecting to the internet is brought by the risks related to the security and privacy of cloud connection.

7.3.1 Security and Privacy in Cloud Computing

Whether people are aware of it or not, issues regarding security and privacy in cloud computing systems and, more in general, the protection of data on the internet are one of the major challenges of the last decades.

Along with the digitaliazion of our society, data have become highly important and valuable. More than ever, in the last years, due to the COVID-19 pandemic, the need of being able to carry-out daily working tasks and routines from a remote location has grown exponentially. Even those who are not aware of the risks of sharing data on the internet are doing so on a daily basis. Moreover, many activities were forced to relocate their businesses on websites and make them *smart*, often exposing their organization to the possibility of loss or theft of sensitive data because of a cyber attack or a data breach.

As mentioned, Amazon, together with its vocal assistant platform, offers these businesses many tools for renewing themselves in a time were digitalization and e-commerce are fundamental. However, exploiting these services without proper awareness and notice can sometimes have very dangerous consequences.

Without going too much into the details of the problem concerning the cloud systems security, since it goes beyond the topics of this work, the main risks behind it are briefly reported in order to explain how relying on a cloud-based vocal assistants is not always the best choice.

Some of the issues related to data and cloud systems are listed below [17] in order to show the extension of the risks related to the usage of these services:

- data and privacy disclosure;
- data destruction;
- service availability;
- external cybersecurity attack;
- accidental information leakage;
- access rights management;
- identity identification;
- multi-tenant and cross-domain sharing.

Being able to exploit a vocal assistant at the edge, therefore running directly on the robotic platform, allows to avoid the mandatory use of an internet connection and therefore assures that the data are always protected, because never shared.

In private environments, as can be homes or businesses and, even more, in hospitals and healthcare centers that deal with highly sensitive data, the usage of machines exploiting, for example, Amazon Alexa can be often questionable.

For these reasons to assure the protection of the privacy and security of data, implementing a local vocal assistant can be a valuable alternative to exploit service robots with vocal commands, without any cybersecurity risk.

Part III

Low-power Vocal Assistant at the Edge

Chapter 8

Machine Learning in Speech Recognition

The innovative machine learning approach widely exploited by AI applications of any kind, is the key point for creating a local vocal assistant for service robots. In this chapter the author gives an overview on ML, especially applied to speech recognition and natural language processing, in order to give the reader the needed knowledge to understand the different parts composing the intelligent agent. Moreover, some applications of these innovative technology are also presented, to underline the importance of heightening their efficiency and accuracy.

8.1 Machine Learning Overview

Machine learning is the subclass of artificial intelligence in which the purpose is not only having a computer able to do what we order it, but also capable of learning how to respond and behave from past experiences. In other words, the objective of this discipline is not building a machine that can act upon a set of programmed rules manually inserted by men, but instead the ability of training that machine and teaching it how to respond even when the set of possible demands is not predefined.

It is important to underline that when talking about machine learning one should not think about the concept of learning as the human ability to use consciousness. Indeed, in this case the machine uses data to simply find statistical patterns on which models used for future predictions are based.

This is why machine learning is particularly useful whenever it is not possible to identify a specific pattern of input data that a problem could require; therefore, in these cases many training data are exploited, in order to automatically improve the ability of a machine to perform a task.

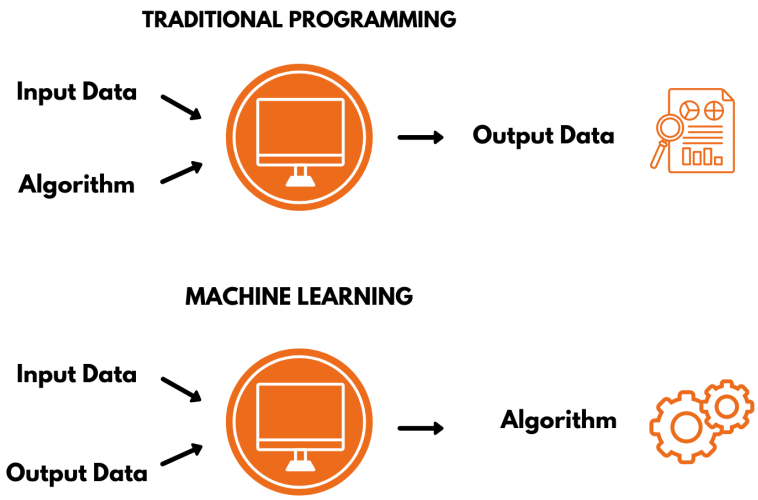


Figure 8.1: Traditional Programming vs Machine Learning.

Nowadays, there are two main branches of machine learning that are mostly used: on one hand gradient amplification algorithms, for shallow learning, and on the other hand deep learning. In particular, the last one became very popular in the last decade as it allowed to solve many challenges and speed up researches in many fields, such as the natural processing language area. However machine learning is a much more vast topic and it is very important not to superimpose one to the other, but on the contrary understand that sometimes deep learning is not the best choice: data could be not enough or a much more simpler algorithm could give the desired result with a lower computational effort.

When talking about any machine learning technique it is important to describe the workflow of the learning process, made up of three steps: training, validation and testing. For any of these steps it is necessary to have a proper quantity of data available, therefore most of the times the exiguous dataset available is the main limit of machine learning, even though there are many techniques to compensate this shortage. The training allows the machine to find a pattern or recognize certain features in the input data, the validation instead allows to check whether the objective function and loss function are improving and eventually tune the weights accordingly. Finally the testing should be done afterwards when the tuning of the parameters is completed, only to check if the model works on a completely new set of data as well. Based on the input data available and the goal required it is necessary to pick the right algorithm for the training process, there exist four main branches making up the taxonomy of machine learning:

- supervised learning
- unsupervised learning
- semi-supervised learning
- reinforcement learning

Supervised Learning: the supervised learning algorithms are those in which the machine does not learn on its own but it needs an external help. In particular during the training phase the feature engineer provides the machine with labelled input-output pair of data. For this reason these kinds of algorithms are often used for classification or regression tasks. The most famous supervised learning approaches are the decision tree, the Naïve Bayes and the support vector machine.

Unsupervised Learning: in unsupervised learning algorithms labels and targets are not used, instead the purpose of the algorithm is to require the machine to autonomously find patterns and transformations in the input data. This approach is often used in the pre-elaboration phase since it requires minimum effort and it allows to have a better dataset to work on with another type of algorithm. In particular feature reductions and K-means clustering are the most famous techniques.

Semi-supervised Learning: semi-supervised learning algorithms involve labels as the supervised ones, however in this case they are not introduced by the feature engineer but by the machine itself thanks to heuristic techniques. Generative models and self-training models are the most common ones in this category.

Reinforcement Learning: reinforcement-based algorithms teach the machine how to behave on the basis of a reward concept. Labels to the input data are omitted, however some rewards are given to the machine during the training according to the choices it makes. In this way the agent that is being trained learns the best way to act in order to maximize the total reward.

8.2 Deep Learning Overview

The adjective *deep*, describing this type of machine learning technique, simply describes the structure of the algorithms: layered and hierarchic. In order to deal with these kinds of architectures it is important to think of the input data as a set of high level features that can be described by putting together lower level features. Indeed, the key-point of deep learning is the use of successive layers as if they are filters through which the input data are passed and broken down. In this

way features at different levels of abstraction can be gradually comprehend by the machine.

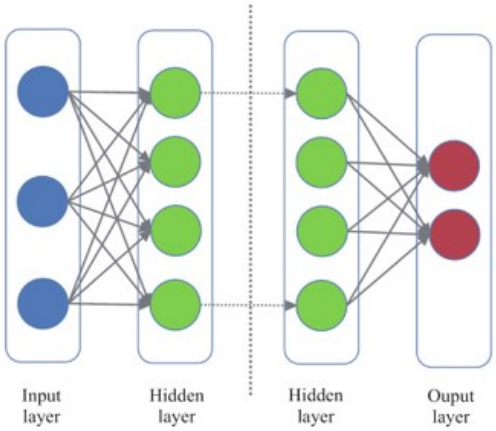


Figure 8.2: Deep Neural Network Basic Structure.

It is thanks to deep learning that the field of machine learning has had a major breakthrough in the last decade. Deep-learning architectures have been used in the last years for many innovative applications such as computer and robotic vision, medical image analysis, speech recognition and in general natural language processing, giving impressive results.

Actually, the ideas and formulations behind the most famous and used algorithms of deep learning were already described in the 90s. However, since this field is mainly based on experiments rather than theory, it wasn't until recent years that it started giving good results. This happened because of previous limitations in the hardware and in the availability of extended datasets. Since in the new century the speed of CPUs and the power of hardware overall has increased exponentially, so have the outcomes obtained by applying deep learning algorithms.

Most of the times these layered structures are artificial neural networks, that can be either exploited for supervised, unsupervised or reinforcement algorithms. The concept of ANN is based on a stratification and that is what makes it suitable to deep learning. The input layer takes the input data and passes them to the hidden layers for processing it, finally the output is given by the output layer. In practice, the processing abilities of a single layer are described by the weights parameterizing it, therefore the aim of deep learning is finding the appropriate values of these weights such that the input data are correctly mapped by the input layer and the hidden and output layers allow to obtain the required targets.

Two important characteristics of deep neural networks (DNN) are the *objective function* and the *loss function* which describe how far the output data are from the target and the overall accuracy of the algorithm. Basically the problem can

be reduced to finding the weights that minimize the objective function, which means that the obtained result is as close as possible to the target one. In many cases the optimization is achieved through back-propagation, in particular the most exploited one is the gradient-descent optimization. Since it is possible to compute the gradient of the loss function, the optimization is done by moving the parameters of the neural network in the direction opposite to it in order to reduce the distance from the target and the value of the loss function.

The main problems caused by the quantity of training data selected are called underfitting and overfitting. During the training process a significant part of the dataset is entered as an input to the machine, so that it can learn from it. At the beginning of this first step the machine shows underfitting, which means that it still did not learn any pattern or elaborate any model from the input data, however even when too many similar data are inserted a problem arises: the machine learns those data too well and can't recognize the new one as part of the same group, this is called overfitting.

8.2.1 Convolutional Neural Network

Convolutional Neural Network (CNN) algorithms are based on convolution, hence the best way to describe this technique is to portray it as a window of a certain dimension sliding over the characteristic input map extracting local pattern instead of the global picture all at once. CNN is made up of two types of layers alternated which are the convolutional layer and the pooling layer. The aim of the convolutional layer is to detect the local conjunctions of features from data coming from the previous layers, while that of the pooling layer is to put together these local features composing a global representation ready to be manipulated by the next convolutional layer [18].

In other words the convolutional layer filters the raw data extracting a certain number of features, then the pooling layer down-samples the data coming from the convolutional one in order to reduce the dimensions and lower the computational effort. Finally a fully-connected layer is needed whenever a classification task must be performed. The training in a CNN structure is usually performed through the most common back-propagation algorithm, the gradient-descent optimization. CNN is mainly used for computer vision because images are made up of features in which the location does not matter and what counts is the feature composition.

Finally this network allows to reduce the computational effort, but not to take into account the sequential features of a sentence or a vocal command. For these reasons CNN in speech recognition and NPL are usually used together with other networks, such as RNN or DBN, for reducing the dimensions of the raw input data.

8.2.2 Recurrent Neural Network

In speech recognition and natural language processing applications recurrent neural network and all the techniques deriving from it usually give the best performances. A RNN peculiarity is certainly the memory feature that allows it to be suitable for processing sequences. While feed-forward networks, such as CNN, can't retain past information, RNN are based on an internal iterative cycle that allows to handle the same data across many layers capturing temporal patterns in sequential data such as speech [18].

Even though RNN are very effective on sequential data, they have a limitation regarding the traditional gradient-descent approach, indeed gradient disappearing may happen. Long-short term memory networks (LSTM) are a type of RNN that solves the vanishing gradient problem, using special hidden layers to keep in memory past information for a longer period of time of the sequence. Using these units, called gates, the network can be trained to retain some features for a longer time and to forget irrelevant ones immediately.

8.3 Audio Processing for Deep Learning

Altough natural language processing and speech recognition algorithms are very important when dealing with vocal assistants, audio processing also plays a key role in this field. Both for key-word detection and speech-to-text translation, necessary for building this intelligent agent, audio files, and usually WAV files, are processed. Moreover, the pre-processing phase of these files is of extreme importance, so that the machine learning algorithms can work properly and efficiently.

This kind of data are simply sound waves that can be plotted as signals in which the amplitude represents the sound intensity, while the period represents time, often translated into frequency.

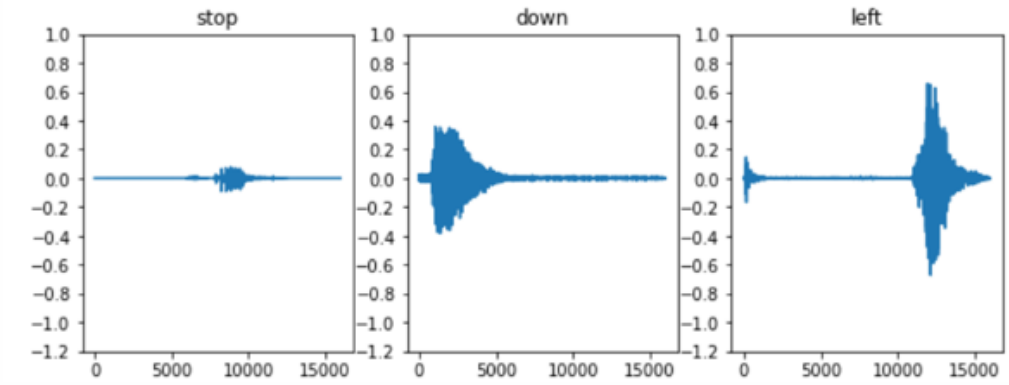


Figure 8.3: Samples of Audio Signals.

For machine learning purposes it is obviously necessary to digitalize these signals, in order to be able to input them into the machine for the training. Moreover, pre-processing of these data is of primary importance for achieving accurate results in this field.

As often happens in the telecommunication field, it can be useful to exploit the spectrum of a signal instead of the signal itself. Indeed, each sound wave can be seen as the sum of many different frequencies and the depiction of the union of all of them can be seen in the spectrum obtained through the Fourier transform of a signal.

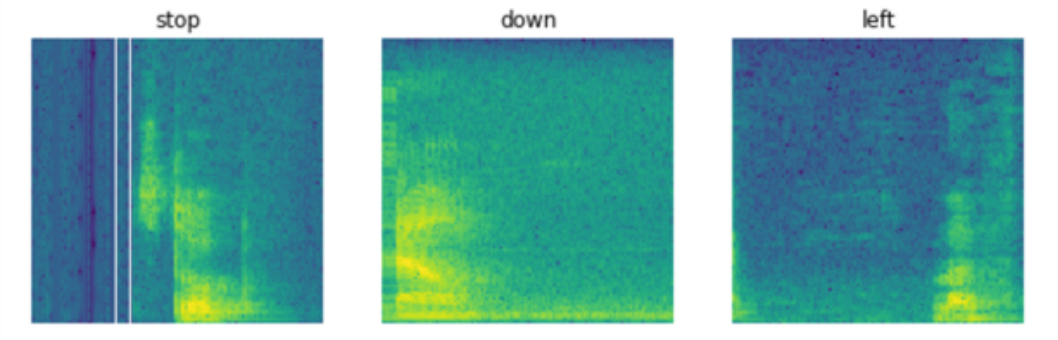


Figure 8.4: Samples of Spectrum of Signals.

Thanks to this 2D visual representation of a signal, in which brighter colors indicate higher energy of the signal, it is possible to train a neural network simply with images, therefore convolutional neural network can be exploited and give good results. It is easily understandable that this approach is better suited for short audio data.

Whenever longer files are processed, in addition to convolutional networks, RNN are often applied to spectrogram images, in order to extract the sequentiality features of audio files as well.

8.3.1 Mel Spectrogram

An interesting aspect to be taken into account when processing human voice, is that the range of amplitude and frequency of the sound used does not cover the whole scales. This results into very dark spectrograms, with few spots of color, giving little information on the sample.

Since human beings perceive both frequencies and amplitude on a logarithmic scale rather than a linear one, it is possible to define a new scale in order to better depict the spectrogram, highlighting the differences between audio samples in an easier way.

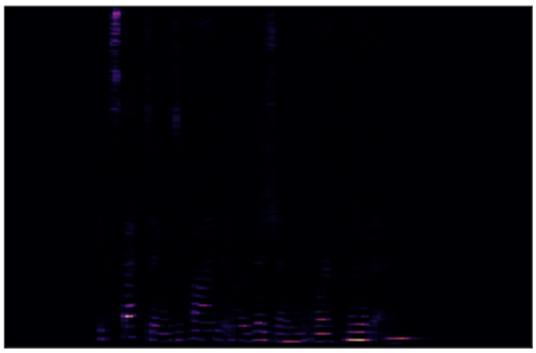


Figure 8.5: Regular Spectrogram Example [19].

The type of spectrograms obtained from this transformation is called *Mel Spectrogram*, which uses the Mel Scale to indicate the frequency and the Decibel Scale to indicate the amplitude, through colors, so that extracting features from the audio sample becomes an easier task.

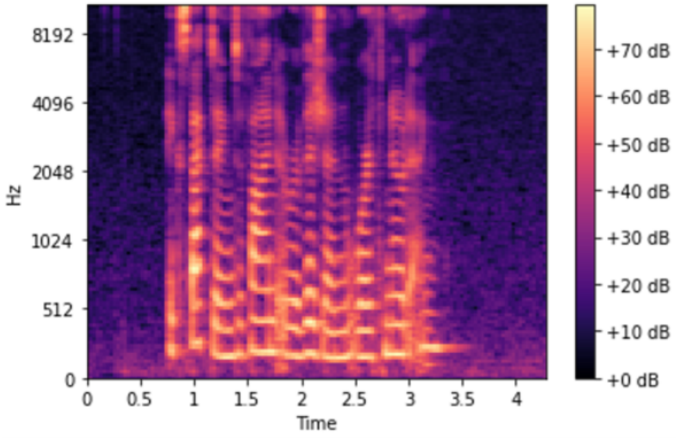


Figure 8.6: Mel Spectrogram Example [19].

Chapter 9

Action Interpreter

Zero is the name of a low-power vocal assistant implemented by a team of researchers at PIC4SeR, the Interdepartmental Center for Service Robotics at Politecnico di Torino. Instead of equipping an indoor service robot with one of the vocal assistants available on the market, which can rely on powerful computational resources and accurate results, the choice was to develop a vocal assistant *at the edge*, therefore running locally on the robotic platform without requiring an internet connection. This decision is based on the desire to overcome the limitations imposed by the use of existing vocal assistants, among which the need of connecting to the internet, and to avoid to expose the user's data to privacy threats.

A local vocal assistant can't be successfully exploited for general purpose applications, as it happens with Alexa, because it is equipped with a limited amount of resources. However, these intelligent agents can be really useful for specific purpose applications, such as vocal navigation in an indoor environment. Indeed, as it happens in this work, the actions the robot can perform belong to a small set, therefore the vocal assistant needs to simply link the given vocal command to one of the actions or, in case it can't classify it, prompt an error. The set of actions can be even wider than the one analyzed in this thesis work; however, in order to have an efficient application, it is necessary for it to be finite in order to exploit supervised learning techniques.

It is important to underline once again that, since Zero is built to be deployed directly on the hardware, the main focus when designing it goes towards creating a lightweight and low-power application.

9.1 Elements of a Vocal Assistant

In the next sections the author will briefly describe an overview of all the different applications composing this vocal assistant. Then a deepening on NLP will follow

since the purpose of this work is to improve the action interpreter of Zero.

The goal is to allow the machine to classify the received textual command, even though the sentence structure used by the user differs from the one defined as action utterance. An improved interpreter allows to make up for mistakes made by the speech-to-text translator as well, understanding the overall meaning of the command even though some errors occur.

Although Zero can be applied to many fields, this thesis focuses on highlighting the differences, the advantages and the disadvantages of using a local vocal assistant rather than Amazon Alexa, therefore Zero will be combined, simulated and tested on the previously presented TurtleBot3 together with the ROS controller node used in the first approach.

Next the author briefly presents the various modules composing the Zero action interpreter, implemented by other members at PIC4SeR, though without going too much into details. Then the action interpreter module, on which this work focuses, will be analyzed more in depth, describing how it was implemented, the advantages of exploiting it and how it was tested and deployed.

9.1.1 Audio Processing

Firstly the application deals with audio files recorded by a microphone: it exploits the Python module PyAudio to record and buffer audio samples. The raw audio is placed into a queue and therefore it can be streamed at low latency.

In addition, after processing the frames of audio samples, the previously presented technique that exploits the conversion to a Mel spectrogram is applied as well. The frequencies to Mel scale are obtained using HTK formula and the overall waveform is converted to a log-magnitude mel-frequency spectrogram.

9.1.2 Wake Up Word

In order to activate vocal assistants, the most exploited way is through a wake up word, indeed this method is a lot faster and simpler than pressing some combination of buttons. For this reason, the recognition of a predefined word is a crucial feature to be implemented. Although it may seem like a simple task, usually the wake-up-word recognition requires to be handled by a limited amount of resources and with an high accuracy, becoming one of the toughest modules to be implemented.

For Amazon Alexa the keyword is its name, *Alexa*, however for opening a skill an additional command needs to be asserted, telling it to open the skill identified by a certain *invocation name*. Therefore, the wake-up sentence to start the vocal assistant for the application becomes very articulated and hard to remember. This reasoning highlights yet another advantage of implementing an assistant at the edge for the purpose.

Since the newly implemented vocal assistant is called *Zero*, the obvious choice for the wake-up word is once again the name itself. The task of hearing every time this word is pronounced is crucial for the application of the intelligent assistant and it is based on the concept of keyword detection.

In order to be able not to miss any of these wake up words, Zero requires the implementation of a small algorithm, consuming little power and computing resources, whose only purpose is to understand whenever someone pronounces the selected word. Zero then continuously runs this module, listening and trying to detect the keyword among different voices, sounds and background noises. The goal is trying to reduce to the minimum the errors due to the recognition of not spoken words and to not detecting all of the spoken wake up words.

9.1.3 Speech-to-Text

After the wake-up word is detected, the main speech recognition algorithm of Zero starts running. It is based on audio processing and translation from speech to text, done thanks to the Python module *Vosk*.

As previously described, in machine learning applications audio files are handled as spectrograms processed according to the needs. Thanks to a suitably trained model, after asserting a vocal command, Zero translates it into a text data, that can be handled by *feature extraction* and NLP algorithms, in order to understand the meaning of the command and pass the information to the ROS node afterward.

9.1.4 Text-to-Speech

Another important aspect implemented in Zero is the vocal response. Indeed, besides completing the requested action, it is necessary to give the user a feedback on whether or not Zero understood the command. For this reason, Zero is able to give answers, prompt errors vocally and guide the person speaking to it through the overall experience of human-machine interaction.

These kinds of algorithms, exploited for speech synthesis purposes, fall into another category of deep learning networks, particularly they are called generative adversial network (GAN). GAN networks allow to *generate* new sets of data with the same characteristics and statistics as those of the training dataset, such as for example sound waves making up voices.

9.1.5 Action Classification

The link between the command assertion, translated into text by the speech-to-text module, and the action itself, is done by what can be defined *action interpreter*

module. Indeed, through meaning extraction methods, it allows to determine which of the available actions the robot should perform, according to the received text.

In particular, the simpler way to do so is by exploiting a JSON file. In this case, similarly to how it is done in the Alexa Skill creation, the developer simply defines a set of utterances the user could say to invoke each action and a set of responses, positive or negative, and stores them in a JSON file. Using this solution can be a good idea in many cases: it is quite effective and, most of all, does not need any particular computational effort, since it is based on a simple equality operator. However, there are some limitations and some aspects that it is possible to improve.

First of all, using a more efficient action interpreter makes up for any error that can occur in the speech-to-text conversion. Due to background noise or other external factors, the translation could miss some letters or even words, therefore comparing the command with the utterances in the JSON file sometimes fails to classify the command in the right action category. Moreover, predicting all of the utterances another person could use when interfacing with Zero, could be a hard task for the application developer. Finally, it is important to take into account that sometimes sentences using completely different words could mean the same thing, while others with the same words re-ordered can have very different meanings.

In order to further elucidate what is the objective of the action interpreter described, an example is reported. Basically, the vocal assistant should be able to classify all of the following sentences in the same action category, identified by the controller node as "CUCINA":

- *Naviga verso la cucina.*
- *Ho fame, mangiamo?*
- *Preparami una pizza.*
- *Vai in cucina!*
- *Cuciniamo qualcosa da mangiare.*
- *Accendi il forno, per favore.*

It is worth noticing that improving the accuracy, in this case of the action interpreter, always comes with an increase of the computational cost. It is important to keep in mind the trade-off between accuracy and resources usage, trying to find algorithms that can be locally deployed and do not need an internet connection to work, while also guaranteeing good performances.

9.2 Natural Language Processing

Natural Language Processing is the branch of machine learning that deals with converting the human language, usually input as textual data, into a machine-readable representation [20].

NLP can be exploited for any application that utilizes text, in different forms, as input data. Hence, it can be applied also, as in the case of this work, in application of speech recognition whereas audio is previously converted into text. For this reason, language processing is a vast area of research and can be subdivided in branches according, indeed, to the application needs, such as:

- information retrieval (IR);
- information extraction (IE);
- question-answering;
- text summarization;
- text translation.

9.2.1 Pre-processing

Often overlooked, when dealing with automatic text recognition, pre-processing is a fundamental step in the machine learning process. Cleaning the input text is necessary in order to reduce ambiguity and duplication [20]. For example, punctuaction and *stop words*, which are frequently used words that do not add any meaningful significance to the context, are usually removed.

Another important aspect of pre-processing is tokenization, which allows to split the data into smaller pieces that the machine can handle more easily. In particular, at this delicate stage the units (characters, words or sentences) passed to the successive steps of text processing are identified.

9.2.2 Vectorization

Word-embedding vectorization plays a key-role in the information retrieval field; often the major effort when dealing with text data is, indeed, the conversion of these input data to vectors. A vector is an array of numbers of a defined dimension, easily handled by machines.

In particular, vectors allow to show the machine obvious relationships between words. Moreover, hidden relationships between words are also automatically revealed, even those the software developer could not even notice. The creation of patterns between vectors form a representation of the language that the machine

can comprehend; in this way when interrogated the assistant can efficiently link words according to many features, such as presence of letters, semantic meaning, sentiment and so on.

The most popular modules available for the task of conversion of words into vectors are:

- *Word2Vec* by Google;
- *Glove* by Standford;
- *FastText* by Facebook.

Cosine Similarity

Cosine similarity is one of the most popular techniques for extracting features from a set of vectors, created during the vectorization phase. It is based on the concept of *measure of similarity* that is done using the *cosine distance*.

Cosine distance is a way to measure the distance between two non-zero vectors, alternative to the more famous Euclidean distance. It can be defined starting from the formula of the dot product and extracting the cosine of the angle between the vectors, obtaining:

$$cosine(angle) = \frac{\mathbf{u} \cdot \mathbf{v}}{norm(\mathbf{u}) * norm(\mathbf{v})}$$

The value of the cosine distance ranges from -1, opposite vectors, to 1, equal vectors, while all the values in between indicate different levels of similarity or diversity of the vectors [21]. This technique is preferred to the Euclidean distance whenever the length of the vectors is not relevant and the interest is towards the relative position of the vectors, one respect to the others.

It is interesting to visualize some examples of the pattern created using this particular measure of distance between vectors. Analyzing the first graph in Figure 9.1, it is possible to notice that the distance between the words *male* and *female* is the same as that between *king* and the corresponding feminine noun *queen*. Similarly, in the second graph, in figure 9.1, it is evident that the distance between different verb tenses of diverse verbs is constant.

Although, cosine similarity measures may appear sufficiently simple and efficient for many application, in the case of the application analyzed in this thesis it has some limitations. Firstly, cosine similarity works very well on single words, but usually when asserting commands multiple words are used. Moreover, the commands used for the indoor navigation of a service robot include words of the same domain, therefore the distance between them is not very significant.

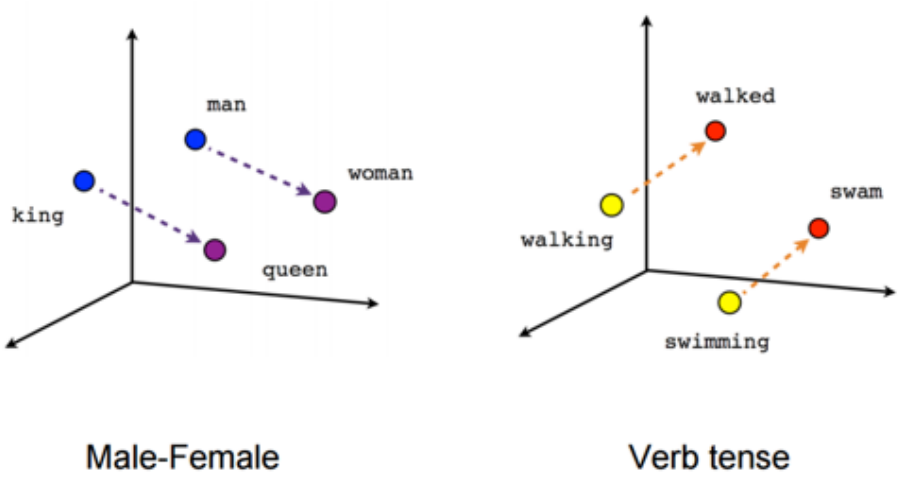


Figure 9.1: Cosine Similarity Relations [22].

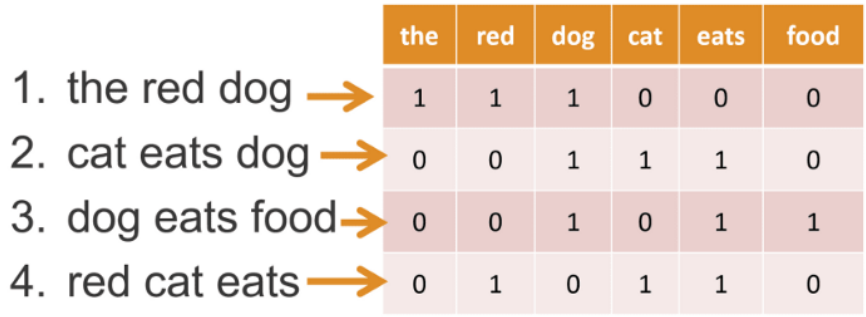
Even though, in the collective imagination, the contrary of *forward* is *backward*, and *left* and *right* are complete opposite words, the cosine distance between them is not very high and therefore this method does not allow the machine to clearly distinct these commands one from the other.

Bag of Words

In order to solve the first problem illustrated, it is possible to look for a way of creating vectors from sentences, instead of single words. *Bag of Words* (BoW) allow to do so exploiting the following strategy.

At first a list of words extracted from a set of text data, which is called vocabulary, is created. Then, the concept behind BoW focuses not on the order in which words appear in a text but simply on whether they do or do not and, sometimes, also with what frequency. Therefore a weight is assigned to each word of the vocabulary, for each sentence analyzed [22], so that a mapping of each word with some kind of feature index is obtained.

This technique has the advantage of taking into account the whole sentence, instead of separated words, however it is not suited for the goal of this work either because it does not take into account the semantic similarity of the sentences. Indeed, the same command assertion could be done with sentences containing all different words. Moreover, this technique would require the availability of a vast corpus of data, which is often hard to retrieve, especially in the Italian language.



	the	red	dog	cat	eats	food
1. the red dog →	1	1	1	0	0	0
2. cat eats dog →	0	0	1	1	1	0
3. dog eats food →	0	0	1	0	1	1
4. red cat eats →	0	1	0	1	1	0

Figure 9.2: Bag of Words Technique Example [23].

9.3 Universal Sentence Encoder

The Universal Sentence Encoder (USE) is a model by Google for encoding sentences into embedding vectors [24]. It allows to convert not only words, but also sentences, or even paragraphs, into embeddings that the machine can process, understand and classify. This last point is crucial since the words handled by Zero are similar to each other, indeed they belong to the same domain of navigation commands. Therefore it is necessary that the algorithm focuses on the overall meaning of the sentences, instead of on the single words. Moreover, researchers at Google proved that sentence embeddings outperform word embeddings in many NLP tasks and they developed the USE model, which results particularly suited for classification, clustering and for transfer learning purposes.

This model is pre-trained on a vast corpus of data, guaranteeing great efficiency and accuracy. Indeed, the main obstacle encountered by machine learning engineers, when dealing with NLP problems, is related to the poor data availability. Exploiting the USE model for transfer learning purposes can create in many cases a simple solution to overcome the problem of the lack of sufficiently large datasets. In details, the main sources of data on which this model has been trained come from Wikipedia, web news and discussion forums.

Commonly, the USE is exploited for extracting semantic similarity features from texts, even though there are many ways to deploy it. When giving in input a string, it allows to output a fixed dimensional representation of the string as an embedding vector. Therefore, the easiest way to deploy the USE for classification purposes is comparing the embedding vectors of many input sentences. In particular, in order to obtain the semantic similarity score between two vectors, the angular distance between the pairwise embedding vectors can be computed as follows:

$$sim(\mathbf{u}, \mathbf{v}) = (1 - arccos(\frac{\mathbf{u} \cdot \mathbf{v}}{norm(\mathbf{u}) * norm(\mathbf{v})})/\pi)$$

Basically, starting from the cosine distance of the two vectors, the angular distance is obtained using the *arccos* function divided by π. This value represents the similarity score which is a value ranging between 0 and 1, where the upper limit represents the complete equality of the sentences.

Moreover, it is possible to plot a matrix of semantic similarity between sentences, based on their embeddings, in order to better understand the correlations among them and picture how the machine creates patterns in the human language. In figure 9.2, the author reported an example of semantic similarity matrix using the multilingual USE [25] model and giving as input some sentences useful for the navigation of the indoor robot.

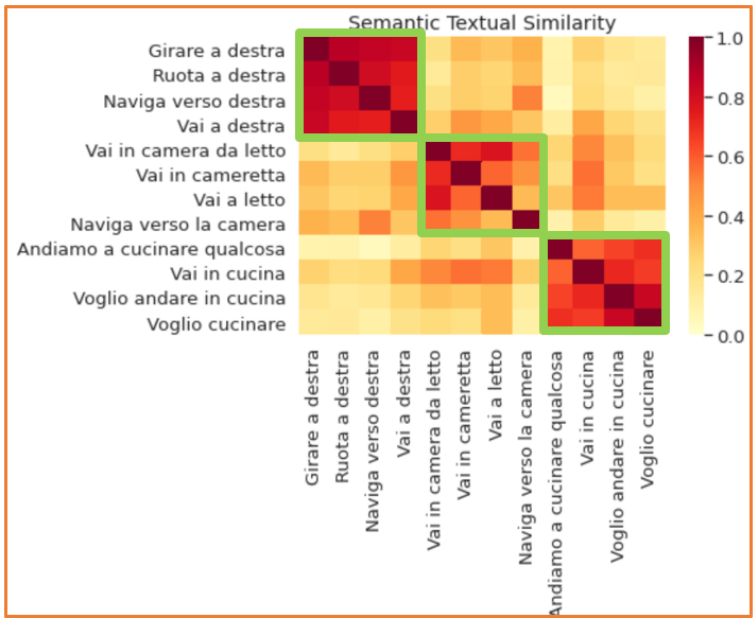


Figure 9.3: Matrix of Semantic Textual Similarity.

The USE is designed as a pre-trained model useful for transfer learning, however its great accuracy allows to exploit it as a standalone as action interpreter of the Zero application. In the case of this work exploiting a USE model allows to make up for a lack of large datasets available, especially in Italian, for NLP applications since a multilingual version of the USE has been developed as well.

Different version of the USE have been implemented. The main distinction is given by a trade-off between accuracy and computing resources; the first model is based on a *transformer* architecture, while the second on a *Deep Averaging Network* (DAN). On one hand the transformer is more accurate, but on the other the DAN

is designed to have higher speed and efficiency. Both models take words, sentences or paragraphs as inputs and output 512-dimensional vectors.

9.3.1 Deep Averaging Network

Deep Averaging Network is one of the two approaches exploited by the USE and it is based on the typical neural network concept.

The intuition behind DAN is that passing the vector average of the input embeddings through a network of feed-forward layers a more abstract and useful representation of the input is obtained. Indeed, these kind of layers allow to extract, at each step, more meaningful information, discarding those that are not important for the word embedding purpose.

More in detail, in the case of DAN this abstraction concept is applied to an unordered composition function, called *Neural Bag of Words* (NBOW). NBOW is a very accurate model exploiting the average of the input word vectors to classify the output, without take into consideration the importance of the single words.

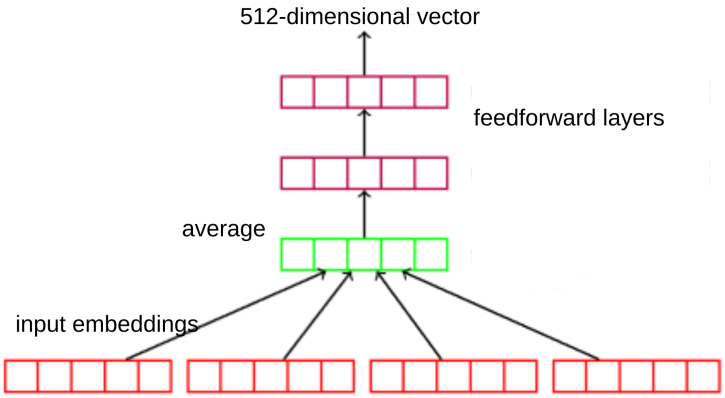


Figure 9.4: DAN Neural Network.

Summarizing, as can be seen clearly in figure 9.4, the levels of a DAN model are the following [26]:

1. vector average of the input embeddings;
2. feed-forward layers;
3. linear or softmax classification;
4. cross entropy loss function;
5. dropout (for increasing accuracy).

9.3.2 Transformer

Transformers are a kind of neural network that recently became very popular among machine learning experts and in particular as useful for sequential data, therefore they are used mainly for NLP applications. Tranformer models have been found to be more efficient than RNN and LSTM models and allowed to solve some issues in older networks, however they are more heavy computationally speaking and therefore more expensive, in terms of time and money, as well.

The architecture of a transformer is based on a encoder-decoder structure. The main idea of this model is based on the concept of *attention* and so aims at weighting the importance of each feature encountered and so at paying attention to the single words, instead of the overall word embedding meaning as for DAN.

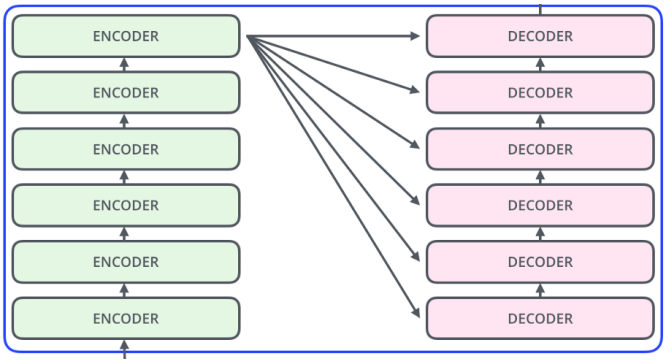


Figure 9.5: Tranformer Structure [27].

As shown, a simplified transformer architecture is made up of the same number (N=6) of encoders and decoders, in which the layer structure is always very similar. Each encoder is usually composed by two layers:

- a self-attention layer;
- a feed-forward neural network;

while the decoders are made up of the same ones, plus an additional attention layer, that allows it to focus on relevant features of the input. Therefore the decoder has the following simplified structure:

- a self-attention layer;
- an encoder-decoder attention layer;
- a feed-forward neural network.

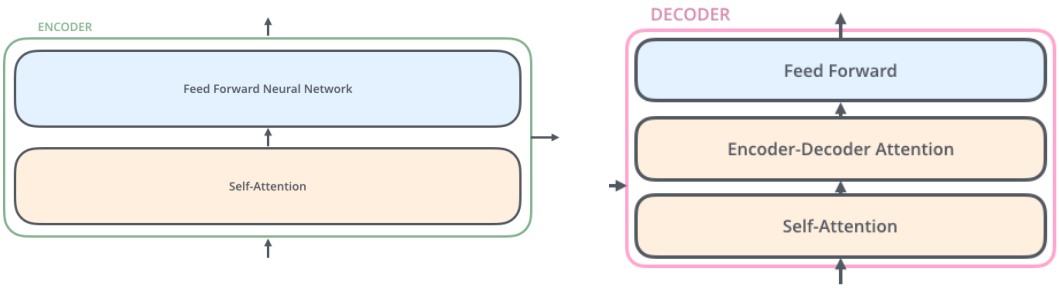


Figure 9.6: Layers of Encoder and Decoder [27].

In detail, the attention module exploited by transformers is called *Multi-Head Attention*, in which the attention mechanism is ran in parallel many times [28]. Afterwards the different outputs are concatenated usually exploiting scale dot-product attention.

9.3.3 Multilingual Universal Sentence Encoder

Even though the DAN-based English USE model already offers a lightweight and quite accurate way to classify the commands asserted to Zero, in order to have a significant growth of the action classification accuracy it is necessary to exploit a model trained on an Italian dataset. Google developed a multilingual version of the USE [25], exploiting a transformer architecture, trained on a large amount of dataset in 16 languages. In particular, Google's translation system has been exploited to assure at least 60 million training data for each language were fed to the algorithm.

The fields of application of the multilingual USE can be very different, for example it is often deployed for translation purposes. Using the multilingual version of the model, the semantic similarity matrix can be built having the words in a first language as entries of the rows, while the entries of the columns can be in a second language.

9.4 From USE to Zero's Action Interpreter

In conclusion, after testing it with many similarity matrices, the author selected the multilingual USE model to be integrated into the application of Zero, serving as *action interpreter*.

In particular, the JSON file previously used for matching asserted commands with available options was kept as sentence database. Then the main idea is the comparison between the embedding of the received command and the embeddings

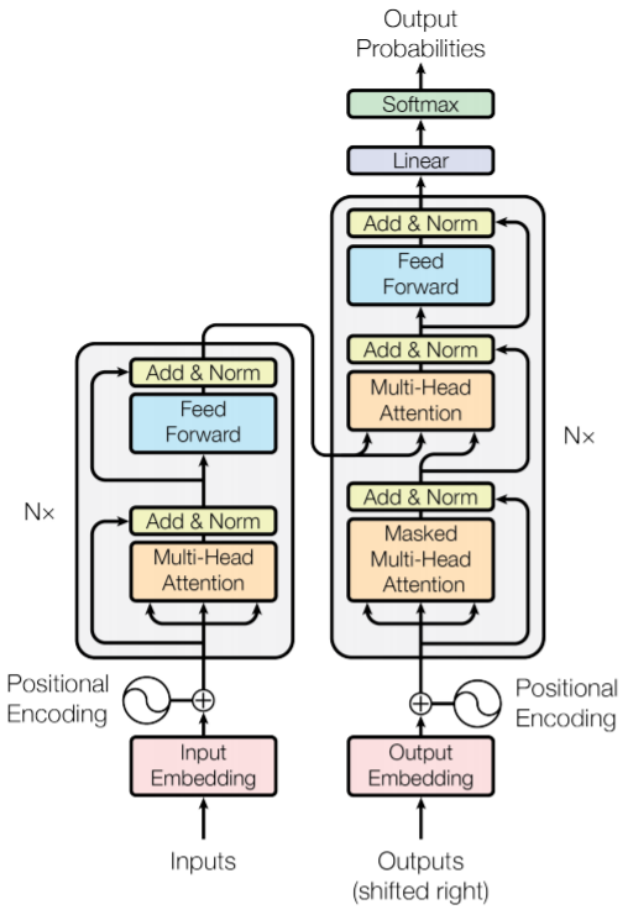


Figure 9.7: Tranformer Overall Model Architecture [28].

of all the possible commands available in the dataset. The higher score of similarity between embedding vectors obtained detects the right command to be executed by Zero. Moreover, a minimum threshold was also set in order to make sure that when receiving a command very different from any of the available, an error message could be sent to the user.

In the main function of the *pic4speech package*, implementing the overall behavior of the vocal assistant, it is possible to find two functions (*action_interpreter* and *action_response*). These functions can be found in the *action_interpreter_USE.py* file, that defines the object class *ActionInterpreterUSE*, that can be found in the Appendix A [3].

An important characteristic of this interpreter to be highlighted, is that it allows to compensate for weaknesses in the other modules of the vocal assistant. In particular, whenever audio processing or speech-to-text are not very accurate, Zero can still understand the action to be executed thanks to the USE interpreter.

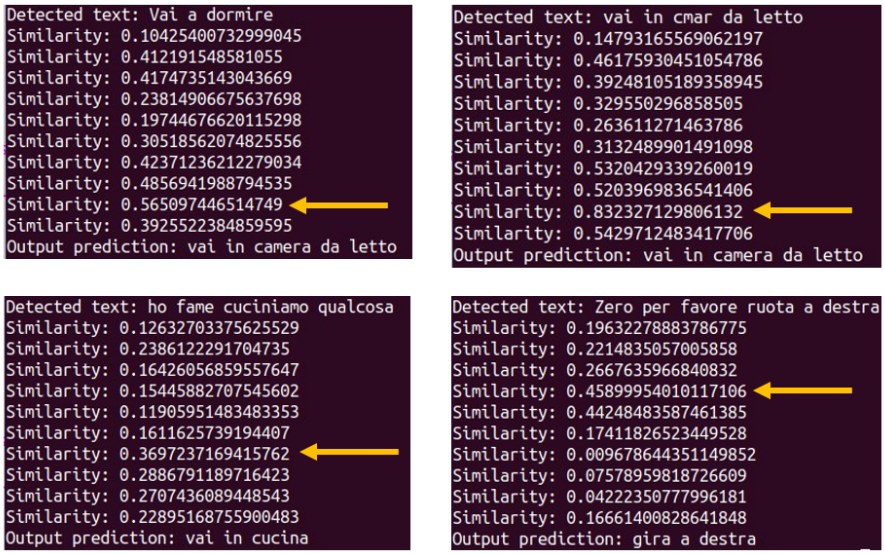


Figure 9.8: Zero's Action Interpreter with USE tests.

As an example, in the top picture on the right in figure 9.8, it is possible to read that the word *"camera"* has been detected as *"cmar"*. While Zero does not have any problem understanding the meaning of the command, if the same error occurs when asserting the command to Alexa, the latest can't understand that the command is telling to navigate to the bed room. Of course, in the case of Alexa the accuracy of the other modules is much higher, however this is an important feature for improving the behavior of Zero and the accuracy of its responses.

Figure 9.9: Alexa vs Zero Test.

Chapter 10

Zero Approach Simulation and Testing

The aim of last chapter of part III is to show the results obtained with the second approach and the advantages brought by exploiting the local vocal assistant. The deployment of the overall application, including the improved action interpreter, is once again done on TurtleBot3. However, it is not possible to deploy the model of the USE on the Jetson Nano because it does not support the *tensorflow_text* package. An intel-based computer, mounted on TurtleBot3, is exploited instead, in order to test the improved vocal assistant.

10.1 Gazebo Simulation

The set up for the simulation of the second approach is similar to the first one, however it does not require an internet connection. As previously described in chapter 7, the simulation is carried on in Gazebo and, more specifically, exploiting TurtleBot3 empty world environment, in which the rooms are marked by some light spots.

The controller node is ran through the *roslaunch* command, as before, however it is not necessary to run the same bridge node for the mqtt connection. Indeed, the vocal assistant is installed directly on the hardware, both for simulation and testing, and therefore the only task of the bridge node is linking the action selected by the USE model to the right action of the controller node.

Once the vocal assistant is launched, it is possible to start asserting vocal commands that are directly translated from speech to text, processed by the action interpreter and finally associated to an action request to the controller node.

In figure 10.1 the author reports an example of exchange of command assertions and answers with Zero vocal assistant. It is important to notice how the USE performs the association between vocal command and action taking into account not only the presence of specific words, but more importantly the whole sentence semantic.

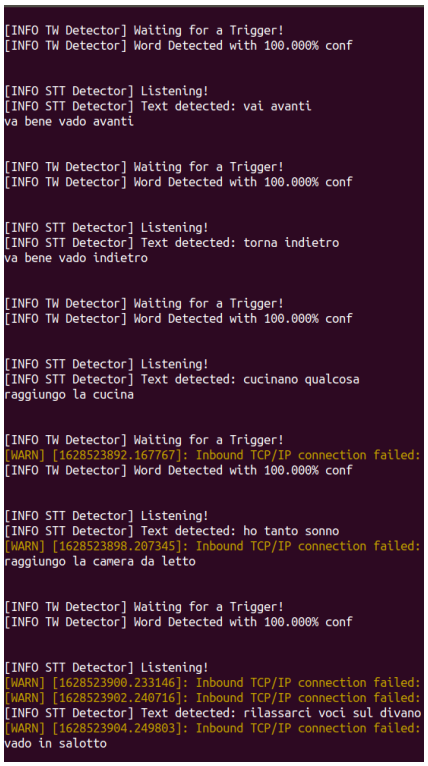


Figure 10.1: Simulation of Zero vocal assistant example.

10.2 Real-time Testing

Some difficulties appear when testing the behavior of the improved Zero vocal assistant on the hardware set up chosen for the previous tests. The main limitation is related to the *tensorflow_text* package, required to run the Universal Sentence Encoder in offline mode. Indeed, the Jetson Nano board does not support this package and so it can't run the model. Since avoiding an internet connection is one of the main reasons for choosing a vocal assistant at the edge, an alternative hardware platform was selected in order to carry out the final test. The choice falls on a small intel-based computer mounted on TurtleBot3 burger, that could be substituted by any intel-based board of smaller size.

Figure 10.2: TurtleBot3 with Intel-based Computer

Running the bridge and the controller nodes on the intel-based device, together with a TurtleBot3 bring-up command, allows to test the overall application. The testing was mostly smooth, however some limitations related to the poor quality of the microphone were encountered. Moreover, the odometry of TurtleBot3 was not very accurate, even though the defined coordinates were reached with good approximation.

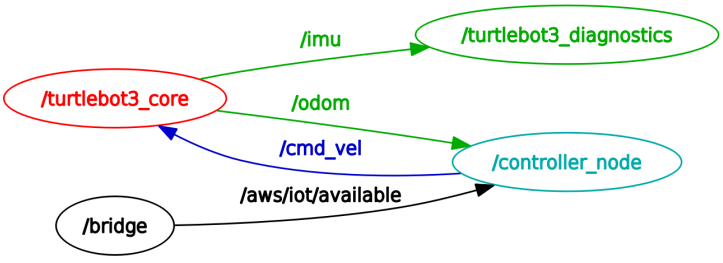


Figure 10.3: Rqt Graph of Testing.

10.3 Advantages

After widely testing Zero vocal assistant with commands aimed at the indoor navigation of a service robot, some conclusions can be drawn on the advantages and disadvantages of using an assistant at the edge.

As expected, the need of satisfying strict requirements related to hardware size and low-power consumption do not allow to reach levels of accuracy comparable with those of vocal assistants that rely on cloud system resources. For this reason, the Word Error Rate (WER), a metric for the measure of the accuracy of speech-to-text application, can't be as low as that of Alexa. The background noise reduction is the main obstacle for reaching good levels of accuracy.

Although there are some limitations related to accuracy and readiness of this system, there are actually many advantages of using it that have been described in the previous pages and are summarized next. First of all, the main advantage of a vocal assistant at the edge, is that it does not connect to any cloud system and so no internet connection is required. This feature guarantees that no interruption of the service occurs because of internet service unavailability or latency, as well as an higher protection of the user from a privacy point of view.

Moreover, another characteristic to be highlighted concerns the meaning extraction algorithm. Indeed, Alexa assures a general purpose assistant, that is capable to comprehend utterances in any semantic field. However, having a model trained on a specific dataset, belonging to a specific semantic area, can be advantageous for having a more precise way of classifying commands asserted by the user. For example, running some tests in parallel on both assistants it is possible to notice that whenever a character in a word is misplaced (e.g. *sinstra*) Zero does not have any problem linking it to the right class, whereas Alexa often fails to match it with the correct action.

Part IV

Conclusion

Chapter 11

Results and Future Work

The goal of this chapter is drawing the conclusions of this work, giving an idea of possible ways to improve it in future researches and finally presenting the applications in which a vocal assistants can help improving service robotics in indoor environments. The author will briefly summarize the characteristics of both approaches to vocal navigation, focusing in particular on the advantages of implementing an intelligent assistant specific for an application.

The analysis and comparison of the two approaches to vocal navigation carried out in this thesis clearly justify the choice of developing a low-power vocal assistant. In particular, all the advantages of developing a specific purpose assistant, instead of exploiting an existing one, have been extensively highlighted and allow to clarify why choosing Alexa or Google Assistant, would impose many limitations.

As previously shown, although the vocal assistant implemented putting together different machine learning models could never reach the computational power and speech recognition accuracy of other cloud-based assistants it can be, on the other hand, perfectly tuned to fit the application in which it is exploited. The effort put into creating a specific purpose intelligent agent is balanced by avoiding the need of an internet connection and of cloud resources, that can in some environments or applications not always be available or easy to exploit.

Furthermore, an important aspect to be taken into account is that of privacy. An implicit advantage of avoiding the support of a cloud system is the complete protection of the acquired data. Indeed, all the vocal commands and background noise heard by Zero, are synthesized and elaborated directly on the robotic platform and do not need to be conveyed on the internet.

Finally, a matter of major importance regards taking into account that the models needed to create the specific-purpose vocal assistant have to be suitable for the selected hardware platform and also be able to work accurately even with a limited amount of resources.

11.1 Testing Summary

In table 11.1 the author listed all of the tests that have been taken out during this project and whether they were successful or not.

ASSISTANT	HARDWARE	TEST COMPLETED
Alexa	Gazebo	YES but connection problems
Zero	Gazebo	YES but poor action classification
Alexa	TurtleBot3 + Jetson Nano	YES but connection problems
Zero&USE	Gazebo	YES
Zero	TurtleBot3 + Jetson Nano	NO tensorflow_text not supported
Zero&USE	TurtleBot3 + Jetson Nano	YES

Table 11.1: Testing Phases.

Although the first version of Zero vocal assistant worked well on TurtleBot3 equipped with Jetson Nano, when deploying the improved Zero vocal assistant, upgraded with the USE action interpreter, the author encountered some problems due to the previously described limitations of the hardware. However, the deployment was successful on the intel-based hardware used to substitute the NVIDIA board.

11.2 Future Work and Applications

In future work, it would be a good investment trying to overcome the hardware constraint, in order to able to deploy the implemented vocal assistant to any kind of processor. Moreover, the work to be done on the whole vocal assistant can proceed in the direction to tune it and make it more accurate for the application.

The major improvements that can be made in this project, related to the Zero vocal assistant are regarding the wake up word recognition accuracy and the background noise detection. Moreover, the action interpreter could be tuned specifically on the small set of commands using transfer learning, however this last step requires to gather a good amount of data to work with.

Moreover, as mentioned when pointing out the working principle of the controller node driving TurtleBot, in order to have a service robot able to actually navigate

any indoor environment listening to vocal commands, other features need to be added. In particular, it is possible to add some ROS nodes to guarantee:

- environment mapping;
- path planning;
- obstacle detection and avoidance;
- more accurate position detection.

The results of this work have been drawn testing the application on a simple robotic platform, such as TurtleBot3 burger, however the main goals were highlighting the salient aspects of vocal navigation and analyzing two different approaches to exploit a vocal assistant in service robotics and this analysis constitutes only the core of a vast variety of applications, especially in service robotics, that can exploit both cloud-based and local vocal assistants.

For example, a *smart wheelchair* [29] equipped with a vocal assistant could be useful for elders or physically limited people. Autonomous wheelchairs are already largely deployed, however, a more straightforward way of guiding them, through voice, could allow a larger group of people to benefit from them.

Other service robots equipped with a vocal assistant could also be a good companion for visually-impaired people and could allow them to become more independent in their homes, helping them finding objects or moving around.

Appendix A

[1] Alexa Skill Lambda Function

```
import random, logging, os, time, json, boto3
from ask_sdk_core.utils
import is_intent_name, is_request_type, viewport
import ask_sdk_core.utils as ask_utils
from ask_sdk_model.ui import SimpleCard
from ask_sdk_model import Response
from ask_sdk_core.skill_builder import SkillBuilder
from ask_sdk_core.dispatch_components
import AbstractRequestHandler
from ask_sdk_core.dispatch_components
import AbstractExceptionHandler
from ask_sdk_core.handler_input import HandlerInput
from AWSIoTPythonSDK.MQTTLib import AWSIoTMQTTClient
from boto3.dynamodb.conditions import Key, Attr
from botocore.exceptions import ClientError
from ask_sdk_model.interfaces.alexa.presentation.apl
import (RenderDocumentDirective,
ExecuteCommandsDirective,
SpeakItemCommand,
AutoPageCommand,
HighlightMode)


# ENDPOINT & CERTIFICATES LOCATION
AWS_IOT_ENDPOINT
= "a2bjbsrv2day9n-ats.iot.us-west-2.amazonaws.com"

VocalbotMQTTClient = AWSIoTMQTTClient("Vocalbot-ASK")
VocalbotMQTTClient.configureEndpoint(AWS_IOT_ENDPOINT, 8883)
VocalbotMQTTClient.configureCredentials('./certs/AmazonRootCA1.
pem','./certs/Vocalbot.private.key', './certs/Vocalbot.cert.pem')

VocalbotMQTTClient.configureAutoReconnectBackoffTime(1, 32, 20)
```

```
33      VocalbotMQTTClient.configureOfflinePublishQueueing(-1)
34      VocalbotMQTTClient.configureDrainingFrequency(2)
35      VocalbotMQTTClient.configureConnectDisconnectTimeout(10)
36      VocalbotMQTTClient.configureMQTTOperationTimeout(5)
37
38      VocalbotMQTTClient.connect()
39
40      sb=SkillBuilder()
41      logger = logging.getLogger(__name__)
42      logger.setLevel(logging.INFO)
43
44      def format_mqtt_message(directive):
45          payload = {}
46          payload['data'] = directive
47
48          return json.dumps(payload)
49
50      def send_mqtt_directive(topic, directive):
51          payload = format_mqtt_message(directive)
52          VocalbotMQTTClient.publish(topic, payload, 1)
53
54
55      # ALEXA SKILL
56      class LaunchRequestHandler(AbstractRequestHandler):
57          def can_handle(self, handler_input):
58              return ask_utils.is_request_type("LaunchRequest")(
    handler_input)
59
60          def handle(self, handler_input):
61              speak_output = "Zero è pronto a partire!"
62              return (
63                  handler_input.response_builder
64                      .speak(speak_output)
65                      .ask(speak_output)
66                      .response
67              )
68
69      class ForwardIntentHandler(AbstractRequestHandler):
70          """Handler for Forward Intent."""
71          def can_handle(self, handler_input):
72              return ask_utils.is_intent_name("ForwardIntent")(
    handler_input)
73
74          def handle(self, handler_input):
75              speak_output = "Vado avanti."
76              send_mqtt_directive("aws/iot/available", "forward")
77              return (
78                  handler_input.response_builder
79                      .speak(speak_output)
```

```
                    .set_should_end_session(False)
                    .ask(" ")
                    .response
            )

    class BackwardIntentHandler(AbstractRequestHandler):
        """Handler for Backward Intent."""
        def can_handle(self, handler_input):
            return ask_utils.is_intent_name("BackwardIntent")(
    handler_input)

        def handle(self, handler_input):
            speak_output = "Vado indietro."
            send_mqtt_directive("aws/iot/available", "backward")
            return (
                handler_input.response_builder
                    .speak(speak_output)
                    .set_should_end_session(False)
                    .ask(" ")
                    .response
            )

    class RightIntentHandler(AbstractRequestHandler):
        """Handler for Right Intent."""
        def can_handle(self, handler_input):
            return ask_utils.is_intent_name("RightIntent")(
    handler_input)

        def handle(self, handler_input):
            speak_output = "Giro a destra."
            send_mqtt_directive("aws/iot/available", "right")
            return (
                handler_input.response_builder
                    .speak(speak_output)
                    .set_should_end_session(False)
                    .ask(" ")
                    .response
            )

    class LeftIntentHandler(AbstractRequestHandler):
        """Handler for Left Intent."""
        def can_handle(self, handler_input):
            return ask_utils.is_intent_name("LeftIntent")(
    handler_input)

        def handle(self, handler_input):
            speak_output = "Giro a sinistra."
            send_mqtt_directive("aws/iot/available", "left")
            return (
```

```
126                 handler_input.response_builder
127                     .speak(speak_output)
128                     .set_should_end_session(False)
129                     .ask(" ")
130                     .response
131             )
132
133     class LocationIntentHandler(AbstractRequestHandler):
134         """Handler for Location Intent."""
135         def can_handle(self, handler_input):
136             return ask_utils.is_intent_name("LocationIntent")(
    handler_input)
137
138         def handle(self, handler_input):
139             location_value = handler_input.request_envelope.request.
    intent.slots['location'].value
140         location_id = handler_input.request_envelope.request.intent.
    slots['location'].resolutions.resolutions_per_authority[0].values
    [0].value.id
141
142             speak_output = "Vado in " + str(location_value) + "."
143         send_mqtt_directive("aws/iot/available", str(location_id))
144             return (
145                 handler_input.response_builder
146                     .speak(speak_output)
147                     .set_should_end_session(False)
148                     .ask(" ")
149                     .response
150             )
151
152
153     class HelpIntentHandler(AbstractRequestHandler):
154         """Handler for Help Intent."""
155         def can_handle(self, handler_input):
156             return ask_utils.is_intent_name("AMAZON.HelpIntent")(
    handler_input)
157
158         def handle(self, handler_input):
159             speak_output = "Zero può andare avanti, indietro, a
    destra, a sinistra o verso una stanza."
160             return (
161                 handler_input.response_builder
162                     .speak(speak_output)
163                     .set_should_end_session(False)
164                     .response
165             )
166
167     class CancelOrStopIntentHandler(AbstractRequestHandler):
168         """Single handler for Cancel and Stop Intent."""
```

```
169         def can_handle(self, handler_input):
170             return (ask_utils.is_intent_name("AMAZON.CancelIntent")(
    handler_input) or
171                     ask_utils.is_intent_name("AMAZON.StopIntent")(
    handler_input))
172 
173         def handle(self, handler_input):
174         send_mqtt_directive("aws/iot/available", "stop")
175             speak_output = "Navigazione di Zero terminata. Ciao!"
176             return (
177                 handler_input.response_builder
178                     .speak(speak_output)
179                     .set_should_end_session(True)
180                     .response
181             )
182 
183     class SessionEndedRequestHandler(AbstractRequestHandler):
184         def can_handle(self, handler_input):
185             return ask_utils.is_request_type("SessionEndedRequest")(
    handler_input)
186 
187         def handle(self, handler_input):
188             return handler_input.response_builder.response
189 
190     class IntentReflectorHandler(AbstractRequestHandler):
191         def can_handle(self, handler_input):
192             return ask_utils.is_request_type("IntentRequest")(
    handler_input)
193 
194         def handle(self, handler_input):
195             intent_name = ask_utils.get_intent_name(handler_input)
196             speak_output = "Direzione non valida. Riprova!"
197             return (
198                 handler_input.response_builder
199                     .speak(speak_output)
200             .ask(speak_output)
201                     .response
202             )
203 
204     class CatchAllExceptionHandler(AbstractExceptionHandler):
205         def can_handle(self, handler_input, exception):
206             return True
207 
208         def handle(self, handler_input, exception):
209             logger.error(exception, exc_info=True)
210             speak_output = "Scusa ho avuto problemi con il comando,
    riprova."
211             return (
212                 handler_input.response_builder
```

```
213                         .speak(speak_output)
214                         .ask(speak_output)
215                         .response
216                 )
217
218     sb.add_request_handler(LaunchRequestHandler())
219     sb.add_request_handler(ForwardIntentHandler())
220     sb.add_request_handler(BackwardIntentHandler())
221     sb.add_request_handler(RightIntentHandler())
222     sb.add_request_handler(LeftIntentHandler())
223     sb.add_request_handler(LocationIntentHandler())
224     sb.add_request_handler(HelpIntentHandler())
225     sb.add_request_handler(CancelOrStopIntentHandler())
226     sb.add_request_handler(SessionEndedRequestHandler())
227     sb.add_request_handler(IntentReflectorHandler())
228     sb.add_exception_handler(CatchAllExceptionHandler())
229
230     lambda_handler = sb.lambda_handler()
```

[2] ROS Controller Node

```
import rospy, time, math
from geometry_msgs.msg import Twist, Point
from std_msgs.msg import String
from nav_msgs.msg import Odometry
from tf.transformations import euler_from_quaternion

# GLOBAL VARIABLES
LINEAR_SPEED = 0.2
ANGULAR_SPEED = 1
LINEAR_ACCURACY = 0.3
ANGULAR_ACCURACY = 0.3
TURN_ACCURACY = 0.1

# INITIALIZE NODE
rospy.init_node('controller')

# PUBLISH VELOCITY
pub_vel = rospy.Publisher('/cmd_vel', Twist, queue_size=1)

# BASIC NAVIGATION FUNCTIONS
msg_vel = Twist()

def stop():
    msg_vel.linear.x = 0
    msg_vel.angular.z = 0
    pub_vel.publish(msg_vel)

def forward():
    msg_vel.linear.x = LINEAR_SPEED
    msg_vel.angular.z = 0
    pub_vel.publish(msg_vel)

def backward():
    msg_vel.linear.x = -LINEAR_SPEED
    msg_vel.angular.z = 0
    pub_vel.publish(msg_vel)

def right():
    sub_odometry = rospy.Subscriber('/odom', Odometry,
odometryCallback)

    goal_theta = theta - math.pi/2
    if(goal_theta <= -math.pi):
        goal_theta = -(goal_theta + math.pi)
    inc_theta = goal_theta - theta
```

```
45
46          while(abs(inc_theta) > TURN_ACCURACY):
47              msg_vel.linear.x = 0
48              msg_vel.angular.z = -ANGULAR_SPEED
49              pub_vel.publish(msg_vel)
50              inc_theta = goal_theta - theta
51          stop()
52
53      def left():
54          sub_odometry = rospy.Subscriber('/odom', Odometry,
    odometryCallback)
55
56          goal_theta = theta + math.pi/2
57          if(goal_theta > math.pi):
58              goal_theta = -(goal_theta - math.pi)
59          inc_theta = goal_theta - theta
60
61          while(abs(inc_theta) > TURN_ACCURACY):
62              msg_vel.linear.x = 0
63              msg_vel.angular.z = ANGULAR_SPEED
64              pub_vel.publish(msg_vel)
65              inc_theta = goal_theta - theta
66          stop()
67
68      # CALLBACK FUNCTION
69      x = 0
70      y = 0
71      theta = 0
72      def odometryCallback(data):
73          global x
74          global y
75          global theta
76
77          x = data.pose.pose.position.x
78          y = data.pose.pose.position.y
79          rot_q = data.pose.pose.orientation
80          (roll, pitch, theta) = euler_from_quaternion([rot_q.x, rot_q.
    y, rot_q.z, rot_q.w])
81
82      # LOCATION FUNCTIONS
83      def mapping(goal_location):
84          goal = Point()
85          if "CUCINA" in goal_location:
86              goal.x = 3
87              goal.y = -3
88          elif "SALOTTO" in goal_location:
89              goal.x = 3
90              goal.y = 3
91          elif "BAGNO" in goal_location:
```

```
92              goal.x = -3
93              goal.y = 3
94          elif "CAMERA" in goal_location:
95              goal.x = -3
96              goal.y = -3
97
98          return goal
99
100     def location(goal_location):
101         sub_odometry = rospy.Subscriber('/odom', Odometry,
    odometryCallback)
102
103         goal = mapping(goal_location)
104         inc_x = goal.x - x
105         inc_y = goal.y - y
106         goal_theta = math.atan2(inc_y, inc_x)
107         inc_theta = goal_theta - theta
108
109         while(abs(inc_x) > LINEAR_ACCURACY or abs(inc_y) >
    LINEAR_ACCURACY):
110
111             if inc_theta > ANGULAR_ACCURACY:
112                 msg_vel.linear.x = 0
113                 msg_vel.angular.z = ANGULAR_SPEED
114             elif inc_theta < -ANGULAR_ACCURACY:
115                 msg_vel.linear.x = 0
116                 msg_vel.angular.z = -ANGULAR_SPEED
117             else:
118                 msg_vel.linear.x = LINEAR_SPEED
119                 msg_vel.angular.z = 0
120
121             pub_vel.publish(msg_vel)
122
123             inc_x = goal.x - x
124             inc_y = goal.y - y
125             goal_theta = math.atan2(inc_y, inc_x)
126             inc_theta = goal_theta - theta
127
128         stop()
129
130     # CALLBACK FUNCTION
131     def driveCallback(data):
132         rospy.loginfo(data.data)
133
134         if "forward" in data.data:
135             forward()
136         elif "backward" in data.data:
137             backward()
138         elif "left" in data.data:
```

```
139             left()
140         elif "right" in data.data:
141             right()
142         elif "stop" in data.data:
143             stop()
144         else:
145         location(data.data)
146 
147     while not rospy.is_shutdown():
148         sub_voice = rospy.Subscriber('/aws/iot/available', String,
    driveCallback)
149             time.sleep(1)
150         sub_voice.unregister()
151 
152     rospy.spin()
```

[3] Action Interpreter with Multilingual Universal Sentence Encoder

```
import json, os
import numpy as np
import tensorflow_text
import tensorflow_hub as hub

embed = hub.load(os.path.dirname(os.path.realpath("utils/model_USE/saved_model.pb")))

class ActionInterpreterUSE(object):
    def __init__(self, config):
        json_file  = self._load_json(config)
        self._get_actions(json_file)
        self._get_responses(json_file)

    def _get_actions(self, j_file):
            self.actions = {int(k): v for k, v in  j_file["actions"].items()}

    def _get_responses(self, j_file):
            self.responses = {int(k): v for k, v in  j_file["responses"].items()}

    def _load_json(self, config):
            json_file = json.load(open(config["PATH_ACT_RESP"], 'r'))
            return json_file

    def action_interpreter(self, detected_text):
        action = embed(detected_text)

        similarity_scores = []
        for i in list(self.actions.values()):
            category = embed(str(i))
            similarity_scores.append(np.inner(action, category))

        if(max(similarity_scores) < 0.3):
            return 0
        else:
            return similarity_scores.index(max(similarity_scores))

    def action_response(self, action_index):
        return self.responses[action_index]
```

Acronyms

AI

Artificial Intelligence

ANN

Artificial Neural Network

ASK

Alexa Skills Kit

ASR

Automatic Speech Recognition

AWS

Amazon Web Services

BoW

Bag of Words

CNN

Convolutional Neural Network

DBN

Deep Belief Network

DAN

Deep Averaging Network

DNN

Deep Neural Network

GAN

Generative Adversial Network

HTTP

Hypertext Transfer Protocol

HMM

Hidden Markov Model

ICC

Instantaneous Center for Curvature

IR

Information Retrieval

IE

Information Extraction

IoT

Internet of Things

IoRT

Internet of Robotic Things

LSTM

Long-short Term Memory

M2M

Machine-to-Machine

ML

Machine Learning

MQTT

MQ Telemetry Transport

NBOW

Neural Bag of Words

NLP

Natural Language Processing

OS

Operating System

ROS

Robot Operating System

RNN

Recurrent Neural Network

SBC

Single Board Computer

USE

Universal Sentence Encoder

WER

Word Error Rate

Acknowledgements

"Things change and friends leave. And life doesn't stop for anybody. I think the idea is that every person has to live for his or her own life and than make the choice to share it with other people. You can't just sit there and put everybody's lives ahead of yours and think that counts as love. You have to do things. I'm going to do what I want to do. I'm going to be who I really am. And I'm going to figure out what that is." -Stephen Chbosky

Se cinque anni fa mi avessero detto che oggi mi sarei trovata a scrivere i ringraziamenti per la mia tesi magistrale in Ingegneria Meccatronica, non ci avrei creduto. Sono certa che gran parte del merito è di tutte le persone che hanno sempre creduto in me anche quando io non lo facevo.

Per prima cosa desidero ringraziare tutti i membri del PIC4SeR, che tra una pandemia e un incendio sono riusciti comunque a creare un ambiente di lavoro allegro e sereno, tra droni, rover e reti neurali.

Un grazie enorme va a coloro che ci sono sempre stati, supportandomi in ogni mia scelta, ma soprattutto sopportandomi: grazie a Mama, Pi e Marta. Vi voglio bene con tutto il cuore.

Grazie a nonna Thea che da piccola mi regalava i lego e i robot senza fare troppe domande e grazie a nonna Titta, con cui a 8 anni decisi che sarei diventata Ingegnere e che vorrei potesse vedermi ora.

Grazie a Enrico, il regalo più grande che il Poli mi abbia fatto. Con lui ho condiviso ogni singolo giorno del mio percorso universitario, dai successi ai fallimenti, passando per tutte le belle avventure in giro per il mondo a festeggiare la fine di una sessione o l'inizio di un nuovo anno. Grazie per essere il mio diario e la mia roccia e per capirmi sempre alla perfezione.

Grazie ai miei amici. Grazie a Luca e Lindsay, che nonostante i molti chilometri che ci separano hanno sempre trovato un modo di essermi vicino. Grazie agli Operativi, i miei compagni di questo viaggio al Politecnico, che tra briscolate e panozzi hanno reso questo percorso impegnativo molto più divertente.

Infine, grazie al Politecnico di Torino che, nonostante abbia provocato numerose crisi di nervi, oggi mi rende fiera di me e del mio percorso.

Bibliography

[1] A. K. Niloy et al. «Critical Design and Control Issues of Indoor Autonomous Mobile Robots: A Review». In: *IEEE Access* (2021) (cit. on p. 5).

[2] L. Sciavicco and B. Siciliano. *Modeling and Control of Robot Manipulators*. Springer, 2000 (cit. on p. 5).

[3] Soonshin Han, ByoungSuk Choi, and JangMyung Lee. «A precise curved motion planning for a differential driving mobile robot». In: *Mechatronics* (). URL: https://www.sciencedirect.com/science/article/pii/S0957415808000512 (cit. on p. 6).

[4] *International Federation of Robotics Website*. URL: https://ifr.org/service-robots/ (cit. on p. 5).

[5] A. B. Nassif, I. Shahin, I. Attili, M. Azzeh, and K. Shaalan. «Speech Recognition Using Deep Neural Networks: A Systematic Review». In: *IEEE Access* (2019) (cit. on p. 10).

[6] URL: https://emanual.robotis.com/docs/en/platform/turtlebot3/overview/ (cit. on p. 13).

[7] URL: https://www.nvidia.com/it-it/autonomous-machines/embedded-systems/jetson-nano/ (cit. on p. 15).

[8] Frank Bentley, Chris Luvogt, Max Silverman, Rushani Wirasinghe, Brooke White, and Danielle Lottridge. «Understanding the Long-Term Use of Smart Speaker Assistants». In: (2018) (cit. on p. 18).

[9] A. S. Sharma and R. Bhalley. «ASR — A real-time speech recognition on portable devices». In: *2016 2nd International Conference on Advances in Computing, Communication, Automationa*. 2016 (cit. on p. 18).

[10] Matthew B. Hoy. «Alexa, Siri, Cortana, and More: An Introduction to Voice Assistants». In: *Medical Reference Services Quarterly* (2018) (cit. on p. 19).

[11] *Total number of Amazon Alexa skills from January 2016 to September 2019*. 2019. URL: https://www-statista-com.ezproxy.biblio.polito.it/statistics/912856/amazon-alexa-skills-growth/ (cit. on p. 20).

[12] *International Telecommunication Union*. URL: https://www.itu.int/en/Pages/default.aspx (cit. on p. 21).

[13] P. P. Ray. «Internet of Robotic Things: Concept, Technologies, and Challenges». In: *IEEE Access* () (cit. on pp. 21, 22).

[14] URL: https://developer.amazon.com/en-US/docs/alexa/ask-overviews/what-is-the-alexa-skills-kit.html (cit. on p. 24).

[15] D. Tosse. *mqtt_ros_aws_iot*. URL: https://github.com/dftossem/mqtt_ros_aws_iot (cit. on p. 28).

[16] URL: https://www.javacodegeeks.com/2016/10/mqtt-protocol-tutorial.html (cit. on p. 28).

[17] PanJun Sun. «Security and privacy protection in cloud computing: Discussions and challenges». In: *Journal of Network and Computer Applications* (2020) (cit. on p. 36).

[18] M. Alam, M. D. Samad, L. Vidyaratne, A. Glandon, and K.M. Iftekharuddin. «Survey on Deep Neural Networks in Speech and Vision Systems». In: *Neurocomputing*. 2020 (cit. on pp. 43, 44).

[19] K. Doshi. *Audio Deep Learning Made Simple*. 2021. URL: https://towardsdatascience.com/audio-deep-learning-made-simple-part-1-state-of-the-art-techniques-da1d3dff2504 (cit. on p. 46).

[20] Amy J.C. Trappey, Charles V. Trappey, Jheng-Long Wu, and Jack W.C. Wang. «Intelligent compilation of patent summaries using machine learning and natural language processing techniques». In: *Advanced Engineering Informatics* (2020). URL: https://www.sciencedirect.com/science/article/pii/S1474034619306007 (cit. on p. 51).

[21] Avishek Garain, Sainik Kumar Mahata, and Subhabrata Dutta. «Normalization of Numeronyms using NLP Techniques». In: *2020 IEEE Calcutta Conference (CALCON)*. 2020 (cit. on p. 52).

[22] P. Pantola. *Natural Language Processing: Text Data Vectorization*. 2018. URL: https://medium.com/@paritosh_30025/natural-language-processing-text-data-vectorization-af2520529cf7 (cit. on p. 53).

[23] C. Breviu. *Grab Your Wine. Its Time to Demystify ML and NLP*. URL: https://www.ronaldjamesgroup.com/blog/grab-your-wine-its-time-to-demystify-ml-and-nlp (cit. on p. 54).

[24] Daniel Cer et al. «Universal Sentence Encoder». In: (2018). URL: http://arxiv.org/abs/1803.11175 (cit. on p. 54).

[25] Yinfei Yang et al. *Multilingual Universal Sentence Encoder for Semantic Retrieval*. 2019 (cit. on pp. 55, 58).

[26] Mohit Iyyer, Varun Manjunatha, Jordan Boyd-Graber, and Hal Daumé III. «Deep Unordered Composition Rivals Syntactic Methods for Text Classification». In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing*. 2015 (cit. on p. 56).

[27] URL: `https://towardsdatascience.com/transformers-141e32e69591` (cit. on pp. 57, 58).

[28] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. «Attention Is All You Need». In: *CoRR* (2017) (cit. on pp. 58, 59).

[29] Andrej Koložvari, Radovan Stojanović, Anton Zupan, Eugene Semenkin, Vladimir Stanovov, Davorin Kofjač, and Andrej Škraba. «Speech-recognition cloud harvesting for improving the navigation of cyber-physical wheelchairs for disabled persons». In: (2019). URL: `https://www.sciencedirect.com/science/article/pii/S0141933119300109` (cit. on p. 68).

[30] F. Chollet. *Deep Learning with Python*. Manning Publications Co., 2017.

[31] R. Haeb-Umbach, S. Watanabe, T. Nakatani, M. Bacchiani, B. Hoffmeister, M. L. Seltzer, H. Zen, and M. Souden. «Speech Processing for Digital Home Assistants: Combining Signal Processing With Deep-Learning Techniques». In: *IEEE Signal Processing Magazine* (2019).

[32] S. P. Reddy Karri and B. Santhosh Kumar. «Deep Learning Techniques for Implementation of Chatbots». In: *2020 International Conference on Computer Communication and Informatics (ICCCI)*. 2020.

[33] S. Sony, K. Dunphy, A. Sadhu, and M. Capretz. «A systematic review of convolutional neural network-based structural condition assessment techniques». In: *Engineering Structures*. 2021.

[34] A. Rajalakshmi and H. Shahnasser. «Internet of Things using Node-Red and alexa». In: *2017 17th International Symposium on Communications and Information Technologies (ISCIT)*. 2017.

[35] *Amazon Developer*. URL: `https://developer.amazon.com/`.