POLITECNICO DI TORINO

Department of Control and Computer Engineering (DAUIN)

Master Degree Thesis

USER INTERFACE FOR AUTONOMOUS DRIVING ROBOT

In partial fulfillment of the requirements for the degree of

Master of Science in Mechatronic Engineering Classe LM-25

Research Supervisor

Prof. Marcello Chiaberge

Candidate

Nicola Bruno

Thesis Advisors

Prof.ssa Marina Mondin

Prof. Fred Daneshgaran

ACADEMIC YEAR 2019-2020

Acknowledgements

I would first like to thank my thesis advisors Prof. Marina Mondin, and the department chair Prof. Fred Daneshgaran for having given me the opportunity to live this incredible experience and for their constant support both educational and psychological during the entire exchange program at California State University, Los Angeles. They were able to create a serene atmosphere in the workplace treating me with respect and professionalism.
I would also like to thank the project tutor as well as CEO and co-founder of the start-up *Innotech Systems* Khashayar Olia for the technical support he gave me during our weekly meetings. I learnt so much about how to develop a fully functioning robot starting from scratch. Moreover, he placed in me its trust, boosting my self-confidence.
Vorrei ringraziare tutte le splendide persone che ho incontrato durante questi anni universitari, dai colleghi, ai professori, a tutti gli amici con cui ho condiviso gioie e dolori, successi e fallimenti.
Ringrazio i miei nonni, che mi hanno sempre sostenuto, hanno sempre fatto il tifo per me e sono sempre presenti in tutti i momenti importanti della mia vita. Ringrazio zia Lilli e zio Raffaele che per me sono stati come due secondi genitori, trattandomi sempre come un figlio e facendomi sempre sentire amato. Ringrazio zia Ninetta, che é stata per me come una seconda nonna, zio Peppe che é sempre stato presente e tutti gli altri zii e cugini. Ringraziamento particolare va fatto poi alla mia piccola grande sorella Gabriella alla quale sono particolarmente legato e della quale sono molto orgoglioso, anche se non lo do molto a vedere. Quando ho bisogno di un consiglio lei c'é sempre e sa sempre cosa dire per tranquillizzarmi.
Infine ringrazio i miei genitori, che hanno fatto tanti sacrifici per non farmi mai mancare nulla e non mi hanno mai fatto sentire un peso nonostante le spese economiche che hanno dovuto sostenere per permettermi di studiare. A prescindere dall'aspetto economico saró eternamente grato a loro per come mi hanno cresciuto ed educato, facendomi capire quali sono i valori importanti nella vita e dandomi consigli senza mai risultare invadenti. Questo traguardo si raggiunge con impegno e dedizione ma anche grazie ad una serenitá mentale che, grazie a loro, non mi é mai mancata.

N.B.

I heard the mission bell
And I was thinking to myself
"This could be heaven or this could be Hell"
Then she lit up a candle and she showed me the way
There were voices down the corridor,
I thought I heard them say
Welcome to the Hotel California
Such a lovely place (such a lovely place)
Such a lovely face.

Abstract

This thesis is about the development of an User Interface for an autonomous service robot operating in an airport environment. The dual objective has been to create an interface at the same time intuitive and beautifully designed in order to be considered by the end user as easy as possible to be faced even for the first time. For this reason an Android based tablet has been chosen for developing the interface, by following material design guidelines for the domain of graphics. The second area that has been treated in this thesis involves the development of a speech recognition algorithm for the implementation of a one-click interface able to minimize the number of steps necessary to perform an action and creating a natural Human-Robot interaction. In this regard, a theoretical analysis about Hidden Markov Models has been conducted, proceeding then with a trade-off analysis that has led to adopt Google Speech Recognizer API. The third domain that has been analyzed in this dissertation is the communication protocol to adopt for the interaction between the various subsystems focusing in particular on tablet-Jetson TX2 MQTT communication and on tablet-airport server communication.
Finally, the last two chapters end the whole discussion. The former describes first the adopted software testing phases for making the algorithm as reliable as possible and then the generic user tests conducted in order to shape the development of the interface on the basis of users' feedback. The latter presents a discussion about possible further implementations, on the basis of the conducted studies.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Summary

Robotics is a multidisciplinary science that deals with the design and the development of technologies able to reproduce as faithfully as possible the human work. Since it is too broad to be discussed as a single topic, the IEEE standard categorized different types of robots that share a variety of similar features. Among all the 15 categories used to classify robots there is the one discussed in this thesis: Consumer or, also called, Service Robotics.
The International Organization for Standardization defines a service robot as:

> *"A robot that performs useful tasks for humans or equipment excluding industrial automation applications" (ISO 8373)*[2].

According to ISO 8373 robots have *"a degree of autonomy"*, which is the *"ability to perform intended tasks based on current state and sensing, without human intervention"*[2]. For service robots the degree of autonomy goes from partial autonomy (a.k.a. the need of a human-robot interaction) to full autonomy (when there is no human intervention). As it will be presented, this thesis will explain the creation of a service robot with partial autonomy, focusing in detail on the design and development of the so called Robotic User Interface. Before starting to develop the project, a preliminary study about what people think about robots has been conducted. Indeed, the constant development of Artificial Intelligence as well as the books and films littered with rogue robots, led some people to be understandably a bit frightened by the prospect of a world overrun by such technologies. Moreover many experts across fields as diverse as technology and economics are also expressing their fears over the rise of the robots in the workplace that could take human jobs over time. So, in the view of all these ethical and social reasons, the project has been developed from both an engineering and a visual appearance point of view in order to create a product that does not scare the end customer. In this introductory chapter an overview of the project will be done, briefly describing the main features of each sub-category that characterizes the whole project and then the adopted system architecture. After that, an insight into the state of the art concerning the existing service robots will be provided trying to take a leaf out of their solutions.

1.1 Overview of the Project

In the last few years the number of industries that decided to invest in the field of service robotics is expanding rapidly. This quick growth is accompanied by strong future growth projections due to the fact that service robots are poised to become a regular part of our lives. According to *TheRobotReport*s robotics review, in March 2019, Venture Capital investments in Autonomous vehicles and robots were more than $1 billion, with $300 million invested in the only field of Service Robotics[3]. These have been the reasons that drove *Innotech Sys.* company to start developing the autonomous driving robot. In the early 2019, the company joined an incubator at San Diego Airport that helps funding the next generation of sharing economy start-ups. The main mission was to develop a robot able to work autonomously into an airport environment and able to help passengers by interacting with them. Among the various tasks, the robot had to be able to guide passengers to the right gate, bring them food, give them useful information about their flight, handle and transport luggages. In order to do that, the project has been split in five main sub-projects:

- Depth-based SLAM
- Object Detection
- Mechanical Design
- Control and Electrical Design
- Robotic User Interface.

Although the main project has been partitioned, each member has worked side by side helping each other and has been updated about the development of each subsection by means of weekly meetings, in order to better understand the general progress of the whole work. In the following sections each sub-project will be briefly explained in order to have a general overview of the robot.

1.1.1 Depth-based SLAM

The term SLAM is an acronym that stands for Simultaneous Localization And Mapping. As the name suggests, the goal of this algorithm is to build a map of an unknown environment and, at the same time, localize the robot so that it can navigate accordingly. SLAM algorithm can be considered the core of the whole project because it makes the robot able to move autonomously. It consists of exploiting the environment to update the position of the robot using LiDAR, cameras or a combination of the two. As far as localization is concerned, cameras, whether monocular or stereo, are at the base of the visual Simultaneous Localization and Mapping (vSLAM) techniques proposed in the literature. Images can be used to save the map of an unknown environment in order to globally position a vehicle or they can be

used to track the motion of an autonomous system by subsequent reference frames transformations, each of them related to the previous state as the state evolves in time. The use of cameras is not limited just to localization, but they can also be exploited to have the perception of the depth and extract important information regarding obstacles or target objects as it will be explained in the obstacle detection section. Last but not least, from the images of the camera the autonomous system can be made able to discern between the nature of different objects using AI techniques and take decisions accordingly [4]. After having performed a case studies, it has been chosen a Stereocamera as our main sensor for the implementation of the vSLAM algorithm.
The term Stereocamera refers to a hardware setup aimed at capturing two images of the same scene from two slightly different point of views in order to exploit the stereoscopic properties. Basically, a stereo system is made up of two identical lenses fixed at a certain distance called baseline. Stereocameras usage rely on the principle of Triangulation, which is the process able to determine the three dimensional position of an object from multiple images, in this case two. The way triangulation works is by calculating the disparity of the two images taken and determining the depth by using intrinsic parameters of the camera such as its focal length, distortion parameters and baseline. Using the ZED camera by StereoLabs 1.1 and its proprietary algorithm ZED SDK 2.8, a real-time depth-based visual odometry and SLAM have been performed so that the error introduced by odometry has been compensated by comparing it with the frame of the corresponding environment.
Indeed, position and orientation of the camera have been computed for every single

Figure 1.1: Zed Camera

frame, creating a full 3D euclidean map. During the tests that have been performed, the robot has been guided in manual mode allowing the creation of a map of the environment. After having stored a database of the actual map, the SLAM algorithm has been launched and, the robot tried to localize itself and at the same time create a map of the environment that was compared with the frames of the map previously created. Other tests have been performed without generating a preceding map but, because of the absence of measuring sensors like, as instance, IMU or LIDARs, the quality of the results was not as satisfactory as the one with the preloaded map. Once the robot localizes itself, starts sending command signals to the Robotic User Interface and to the the F28069-LAUNCHXL evaluation board that in its turn sends command signals to Robot Control Units, allowing the robot to navigate autonomously.

1.1.2 Object Detection

Object detection is one of the fields in which artificial intelligence is in greatest expansion. For years the recognition and categorization of images have been a problem, especially considering the difficulty of a traditional algorithm in recognizing the same object in different positions and angles. Even if it can be considered an easy task for the human mind, for as regards the robot, is not so obvious. Moreover, today's needs go far beyond the simple classification or localization in static images, today the need is to have analysis in real time and, obviously, this goal increases even more the complexity of the problem. The solution is thus to use Yolo, an algorithm that analyze all the parts of the image in parallel, simultaneously, avoiding the use of the sliding window. Yolo was developed by Redmon and Farhadi in 2015. The concept is to resize the image so as to obtain a grid of squares. Then the algorithm makes predictions on 3 different scales, reducing the images of 32, 16 and 8 respectively, in order to remain accurate even on smaller scales. For each of the 3 scales, each cell is responsible for predicting 3 bounding boxes, using 3 anchor boxes [5]. For a matter of corporate privacy, additional details on how the algorithm has been implemented and integrated on the robot can not be provided. However, in general, it can be possible to say that Yolo algorithm is used for recognizing a passenger, so that he can be easily reached for being provided assistance.

1.1.3 Mechanical Design

The mechanical design has been developed with the goal of obtaining a mechanical structure that maximized robot controllability and reliability being at the same time beautifully designed and practical for the different tasks the robot has to accomplish, like deliveries or handling and transportation of luggages. The first problem that has been tackled, was the choice of the wheels configuration. After a case studies it has been chosen to adopt a differential drive configuration with two drive wheels and two castor wheels. The differential drive is a drive system in which there are two active, non-steering wheels with a common axis of rotation and one or more passive castor wheels that have the role of support, keeping the robot statically balanced. This kind of configuration allows the robot to go straight by providing the same actuation to both the motors connected to the wheels. In case of curve it allows the robot to curve by providing different actuation commands to the motors attached to each wheel so that the obtained angular velocities are different. With this kind of configuration is even possible to perform turning in place, by providing to the wheels the same angular velocities with opposite directions [6].
Considering l the baseline, ρ the curvature radius and ω the angular velocity, the

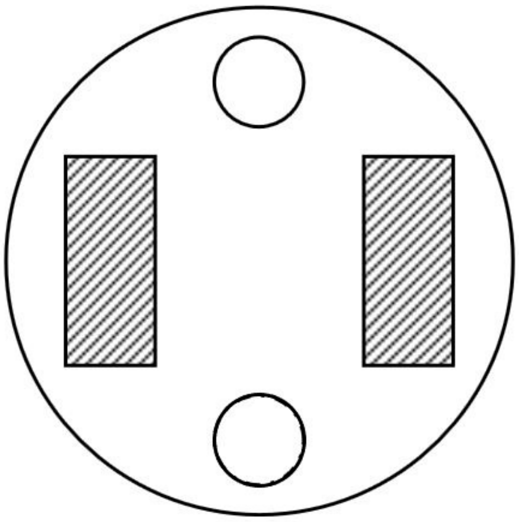


Figure 1.2: Differential Drive Configuration

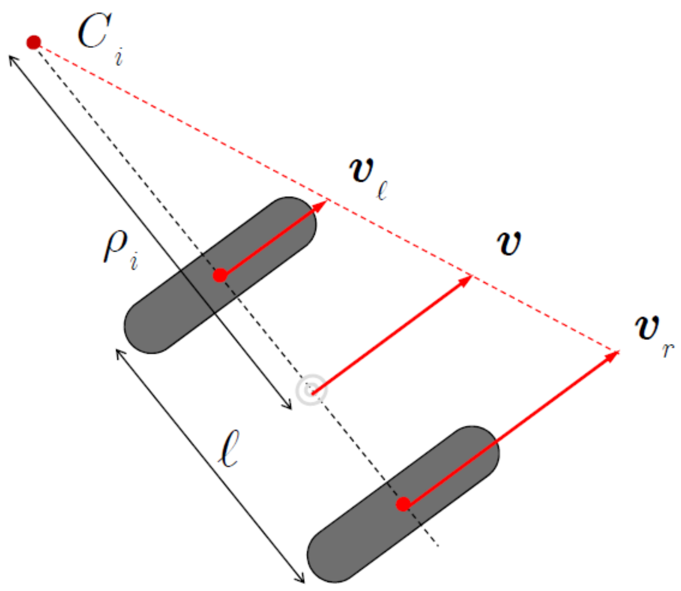


Figure 1.3: Differential Curvature

values of the right and left tangential velocities can be easily computed:

$$v_L = \omega(\rho - \frac{l}{2})$$

$$v_R = \omega(\rho + \frac{l}{2})$$

This kind of situation can be identified as an Inverse Kinematics problem, where after having computed the two tangential velocities it can be obtained the value of the actuation that motors have to provide in order to make the wheels running at the reference speed. Despite its simplicity, the controllability is rather hard to obtain, because the differential drive configuration imposes the Non-Holonomic constraints, that is to say that the number of degrees of freedom of the robot is bigger than the controllable degrees of freedom. So, as instance, it is not possible to perform lateral motion along the wheel axle [7]. There are several reasons why the differential drive choice has been made, but they can be reduced to only two. The former is that every wheel configuration except for omnidirectional wheels configuration introduces the Non-Holonomic constraints so as a matter of mechanical simplicity, differential drive configuration has been chosen. The latter was purely economic, in fact this kind of configuration allows to use two motors instead of three, reducing the cost of the final product.
The second problem that has been deeply analyzed was the choice of the motors, that fell on brushless ones. They have a cost that is higher than a normal DC motor but they are approximately 10% more efficient since they don't need maintenance and they have an higher power density. After having chosen the motor it has been necessary to add a gearbox since speed and torque provided by the motor were not suitable for the application. The third and last problem was the one related with the design, that has been chosen after having considered the state of the art of other robot companies as well as surveys that have been presented to random passengers of the airport and usefulness of the product.
For a matter of industrial secrecy it is not possible to show the final version of the mechanical structure. It presents rounded shapes with the space for the tablet at the level of the robot's head so that the RUI is at the right height to be used properly at the same time conferring to the entire structure a friendly form. Bearing in mind that the mechanical design took months to be developed and that at the same time there was the necessity of running as many tests as possible, a simple very low-cost prototype has been adopted while waiting for the actual chassis. As it is possible to see in figure 1.4, a computer case has been adopted as chassis and a system with two front active steering wheels and two rear active non-steering wheels has been chosen. The prototype mounts four DC motors, one for each wheel, and a stepper motor mounted on the front axis to let the robot turn. All the choices that have been made on the prototype have been dictated by a tight budget as well as a restricted timing so that a rigorous mechanical analysis has not been done. The first step of the development process has been done with the only purpose of testing SLAM and Robotic User Interface as well as their interaction, no matter the type of mechanical structure on which the hardware was mounted.

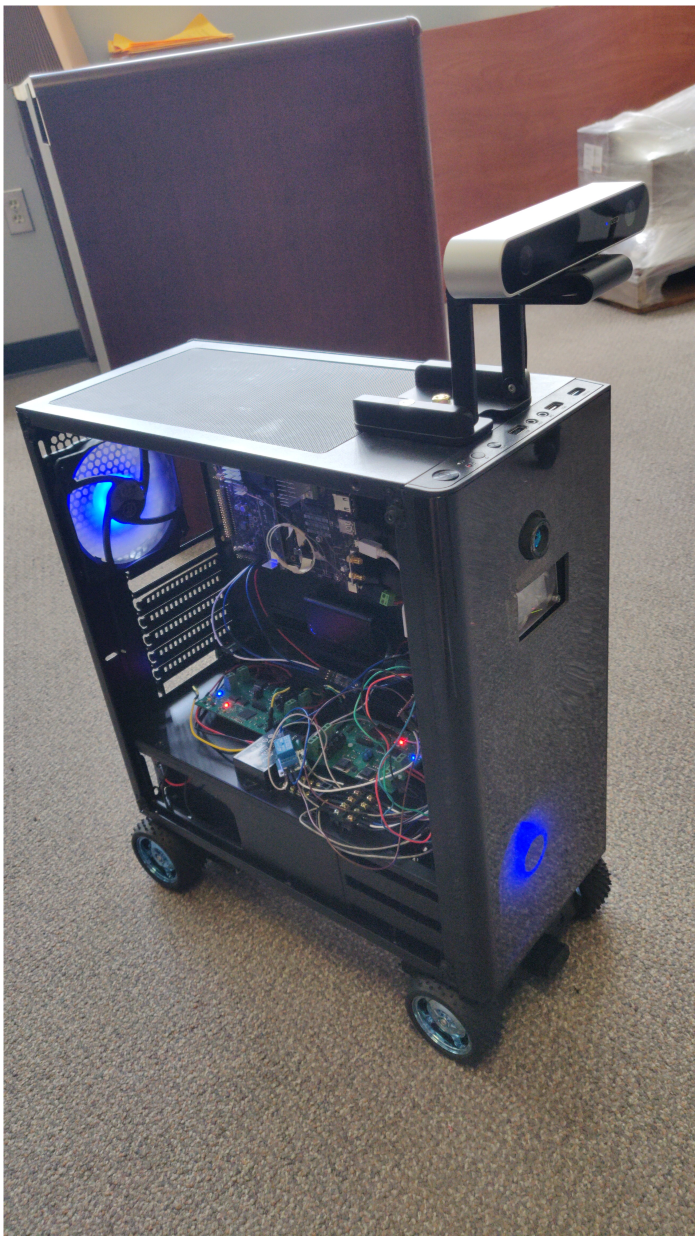


Figure 1.4: Robot Prototype

1.1.4 Control and Electrical Design

The control system design has been done with the final purpose of generating the time functions of the actuating torques/forces, such that the robot motion follows a specified task in the 3D euclidean space. This last is generated by the visual SLAM algorithm, fulfilling specifications on transient and steady-state response. The initial idea that has been considered, was to create a brand new board that included all the components needed for the specific purpose. The list of components includes the MCU of the F28069-LAUNCHXL evaluation board, four full H-bridges, a CAN tranceiver, a booster pack for the CC3220 Wi-Fi Module, CAN channels, a RS232 serial connection, I/0 channels both digital and analog, an ethernet connection (TCP/IP) for communicating with the NVIDIA Jetson TX2 board directly over broadband and a low power microcontroller, able to wake-up all the components via interrupt. This idea has been set aside because of the time constrains and the production costs arising from the necessity of a standardization ISO, so all the electrical components have been mounted separately and wired through CAN or RS232 as will be shown in the following pages. The Robot Control Units have been mounted and wired to the F28069-LAUNCHXL evaluation board through CAN wires and consist of a total of four full H-bridges that allow bi-directional control of the motors (Figure 1.5). The

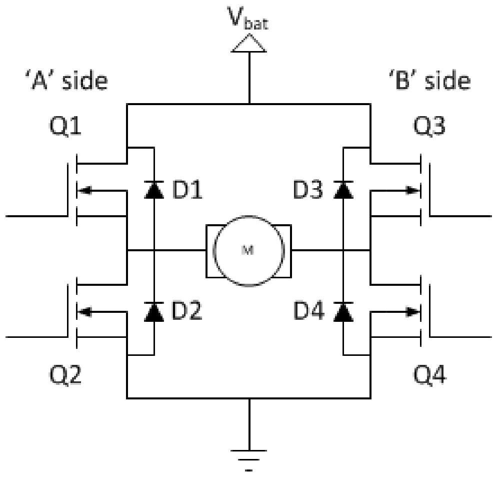


Figure 1.5: H-Bridge Circuit Topology

switching elements are usually bi-polar or FET transistors or in some high-voltage applications IGBTs. In the specific case of this project FET transistors have been adopted. The two transistors below(Q2 and Q4 in the figure) are called *"low side switch"*, since they absorb current from the load; the two transistors above(Q1 and Q3 in the figure) are called *"high side switch"* since they are directly connected to the battery voltage. Depending on which transistors are enabled, there are different possible paths for the current and, as a direct consequence, different effects on the motor behavior as shown in the table below:

Table 1.1: H-Bridge possible combinations

Combination 'A' side	**Combination 'B' side**	**Effect**
0	0	Motor stops
1	0	Motor works forward
0	1	Motor works backward
1	1	Motor stops

Since conditions in which the motor stops can be very dangerous, an inverter has been added to the circuit topology. The wheels have been controlled by two full H-bridges, one for the right front and right rear wheel, the other for left front and left rear wheel, obtaining a system of four active wheels. In order to let the front wheels turn, a stepper motor has been adopted and an incremental encoder has been mounted on it, creating a position feedback control. The SLAM algorithm sends a CAN message with the degree the robot should turn in order to follow the right path. The actual position measured through the encoder is then compared to the desired position and a correction is eventually made. The actual position of the wheels with respect to the reference one is computed counting the edges of the generated square wave that is proportional to the motor rotation. Thanks to the double channel, it has been possible to understand even the direction looking at the phase relationship between the 2 channels. As it's possible to notice in Figure 1.6,

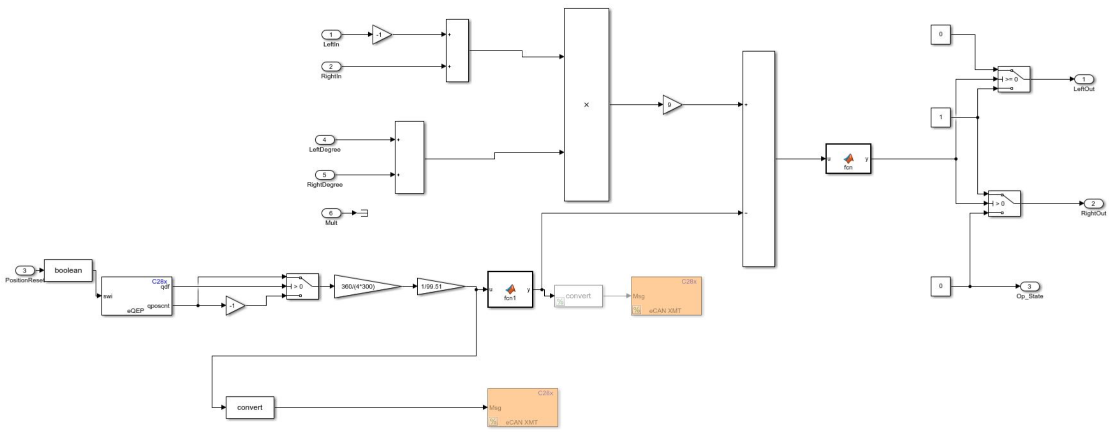


Figure 1.6: Control Scheme

the information about the measured angular position has been computed through the enhanced quadrature encoder pulse (eQEP) Simulink block that has been configured by means of a pin assignment. Before entering in the sum block, in order to be compared with the reference angular command sent by the SLAM algorithm, the counter value is scaled according to the following formula:

$$\alpha = \frac{EdgeCounter \cdot 360}{4 \cdot Res}$$

where the variable *EdgeCounter* corresponds to the actual number of edges detected during the motor rotation, while the variable *Res* corresponds to the resolution of the encoder. The value of α is then further modified depending on whether the motor is turning right or left. If, as instance, the robot is turning right the angle provided by the *eQEP* block is consistent, but in case the robot is turning left the angle has to be subtracted to 360 in order to obtain the same absolute value. Then if the difference between the two values is outside a certain range, it means that robot has to turn in one of the two directions, otherwise it means it is correctly following the path highlighted by the SLAM algorithm. Thanks to this implementation, the robot has been able to complete a delivery during one of the demo sessions at San Diego Airport, by performing a path consisting on a straight, followed by a left turn and a turning in place to come back to the starting point.

1.1.5 Robotic User Interface

As already announced, this thesis deals with the Robotic User Interface sub-project more in detail, focusing on its main features. Using the Integrated Development Environment *Android Studio*, it has been developed an application through which the robot is able to communicate with the end-user, with the airport server and with the Jetson TX2, the board where SLAM algorithm has been implemented. As regards the human-robot interaction, the mission has been to create an interface that was as simple as possible, intuitive and beautifully designed. For these reasons a specific study has been performed in Material Design, focusing on the position, the shape and the illustrations that each icon or button had to have in order to not lead to confusion for the customer.
In order to simplify to the maximum the front-hand, a speech recognition algorithm has been implemented, able to minimize the number of actions that the end user has to perform to accomplish the task. Before implementing the final algorithm a case studies has been conducted, trying to analyze the benefits and the drawbacks of Hidden Markov Models, on which most of the speech recognizers are based.
As regards the inner communications, the mission has been to establish a communication as fast and cheap as possible, in order to not affect the cost of the final product. After having analyzed the state of the art on this field, a MQTT protocol has been selected so that the tablet has been able to communicate with the Jetson TX2 board wireless, with the only necessity of a modem device able to act as a broker during the communication. Regarding the communication with the airport server, a SSH remote connection has been established in order to grant access in a safe way to useful information about as instance the scheduled flights. In the following chapters, each of these aspects will be fully dealt with, analyzing them from both a theoretical and a practical point of view.

1.2 System Architecture

In order to have a clear insight of how the whole system works, it is necessary to understand the communication between the various components. Figure 1.7 shows the System Architecture, briefly analyzing how the components are connected together. In order to better understand how the whole system works, a typical op-

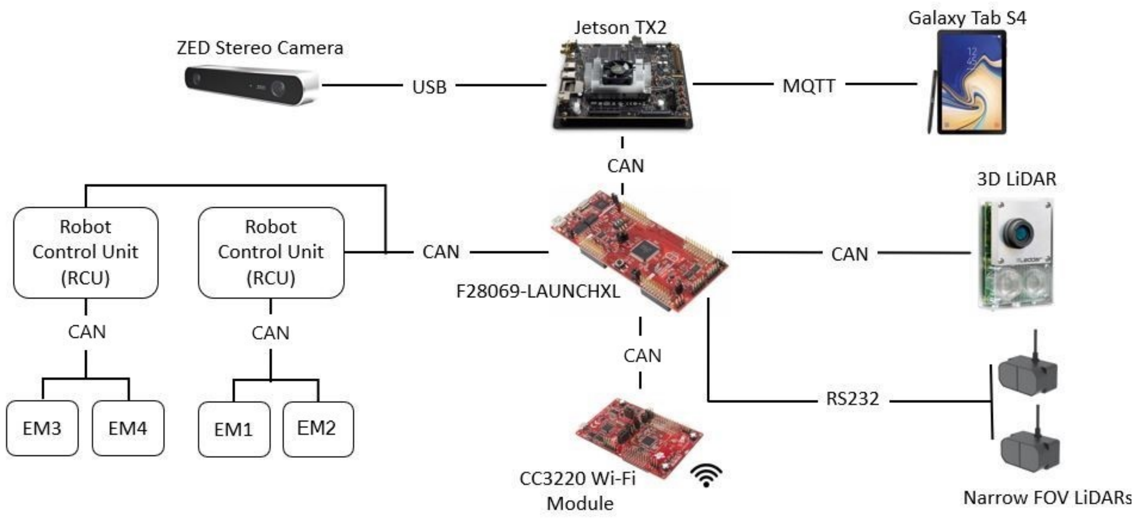


Figure 1.7: System Architecture

erative scenario will be analyzed. Thanks to YOLO algorithm and exploiting the ZED Stereo Camera the robot is able to recognize a passenger, so that he can be easily reached for being provided assistance. The end user interacts with the robot through the interface represented by the tablet. Whenever the user expresses its wish to go, as instance, to a certain gate, the tablet will send a message to the NVIDIA Jetson TX2 board by means of the MQTT protocol. This latter can work thanks to the CC3220 Wi-Fi Module that provides an intranet wireless connection for the messages exchange. As soon as the robot has loaded the map and is ready to go, the Jetson board sends a confirmation message to the tablet and a series of update messages about the navigation so that the user is informed about how far the robot is from the final destination. In order to start the navigation, the Jetson board sends CAN messages to the F28069-LAUNCHXL evaluation board that, in turn, sends CAN messages to the Robot Control Units, responsible for the Electric Motors operations. Each Robot Control Unit is characterized by two H-Bridges, a circuit topology already analyzed in the previous sections. During the navigation, it can happen that some mobile obstacles like people occur. For this reason a 3D LiDAR and two Narrow Field-Of-View LiDARs have been added so that, as soon as an obstacle has been detected, they send messages to the F28069-LAUNCH XL evaluation board that commands the electric motors to stop. After having arrived to the drop point, the Jetson TX2 board sends a MQTT message to the tablet that informs the passenger that the destination has been reached. After that the robot comes back to the starting point, ready to assist other passengers and restart the very same procedure from the beginning.

1.3 Existing Systems

As pointed out by a case studies on robotics from *Silicon Valley Robotics*, in the last few years a lot of new companies entered the world of service robotics trying to get a share of the market. Analyzing the birth and the growth of all these companies, it can be highlighted that all of them focus on a single task trying to develop it as well as possible. Just after the company gets a share of the market it tries to expand from there[8]. With this in mind, after having chosen the airport passengers support as the main field on which being focused, it has been done a thorough research among all the companies that developed a similar kind of service robot, focusing in particular on their hardware components and on their industrial development model.
The first company that has been analyzed was *Savioke* that, with its product *Relay*, represents the state of the art for service robotics. This company decided to create an autonomous delivery robot for an hotel environment able to deliver items to hotel guests. It is characterized by a strong integration with the hotel facilities like hotel's phone system to alert guests when delivery is ready or hotel floors and elevators map to navigate autonomously in the hotel environment. Relay is indeed equipped with cameras and sensors to detect people and objects while navigating, laser sensors to know where it is in its internal map of the building and bumpers to detect when Relay comes into contact with an object or a wall.
The robot is approximately 3 feet tall, weighs less than 100 lbs., has a carrying ca-

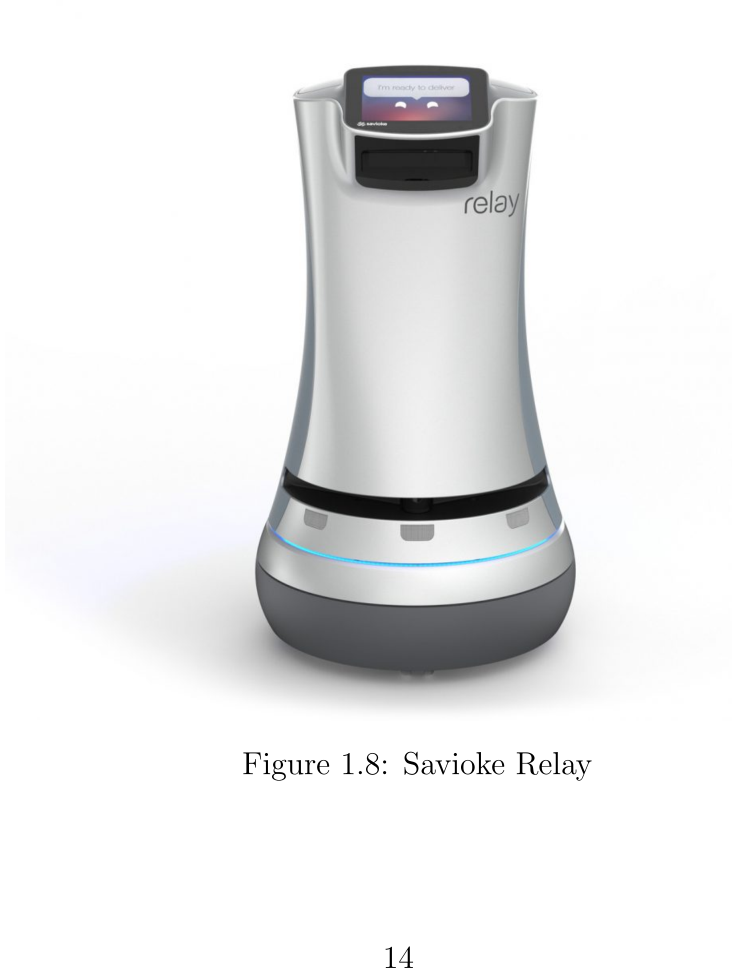


Figure 1.8: Savioke Relay

pacity of 2 cubic feet, and is designed to travel at a human walking pace. Front desk staffers can send Relay out to deliver guest requests and as soon as Relay arrives at guest room, it phones the guest to announce its arrival, delivers the requested items and makes its way back to the front desk [9]. It is equipped with two interfaces, one touchscreen interface and a ROC(Robot Operations Center) interface. The latter is used by the user for monitoring progress, viewing status information or simply for contacting the company in case of issues. Studying in detail this company, it has been decided to adopt a vertical mechanical structure to facilitate the stowage and also to create a strong integration with the airport facilities in order to increase the feeling of safety and efficiency.
The second company that has been analyzed was *Brain Corp* with its proprietary Operating System called *Brain OS*. This company, differently from Savioke, decided at first to focus on AI technologies for robotic floor before entering in the autonomous deliveries field. Brain did not build robots but instead focused on the operating system, which uses artificial intelligence and computer vision to turn existing human-operated devices into autonomous ones. In this way the company obtained the effect of scaling robotic products, providing just the cloud-connected operating system, the *"Brain"* of the entire robot. Studying this company, and in general the majority of robotic companies specialised in software, it has been chosen to employ the NVIDIA Jetson TX2. From its datasheet it can be read that it uses a Pascal-based GPU, as well as two 64-bit Nvidia quad-core ARM chips, with 8GB of RAM on board and 32GB of fast flash storage. It also features built-in Wi-Fi networking, Bluetooth connectivity and 1GB Ethernet for wired connections. This kind of hardware makes it possible to run real-time algorithms like depth-based SLAM or YOLO that can more accurately do things like identify objects in images, create a 3D Euclidean space or interpret surroundings for autonomous navigation.
The third company that has been studied was *Aethon* with its robot *TUG T3*. That company too, it decided to focus on deliveries specializing on heavy loads, with its 1200 lbs. of maximum carrying capacity, 10 hours of battery run time and a torque of 156 in-lbs [10]. TUG T3 has been designed as a smart autonomous mobile robot able to navigate without infrastructures, thanks to an overlapping system of laser, sonar and infrared sensors. The company develops both robots and software control systems allowing their robotic platform to achieve level 4 of autonomy. Even in this case, the robot has been developed with a touchscreen interface, and this brought Innotech Sys. company to go for this kind of interface too.
The fourth company that has been examinated was *Knightscope* with its robots *K1, K3, K5, K7*. This company decided to develop security robots combining Self-Driving Technology, Robotics and Artificial Intelligence. K1 is a stationary machine for use at ingress and egress points, K5 is primarily an outdoor autonomous robot, K7 is an autonomous data machine, while K3 is an indoor machine, so obviously among all the models, K3 is the one that has been studied more in detail.
Equipped with an eye-level 360 Degree HD Camera for streaming and recording, face recognition, human-to-human voice interaction and Automatic Signal Detec-

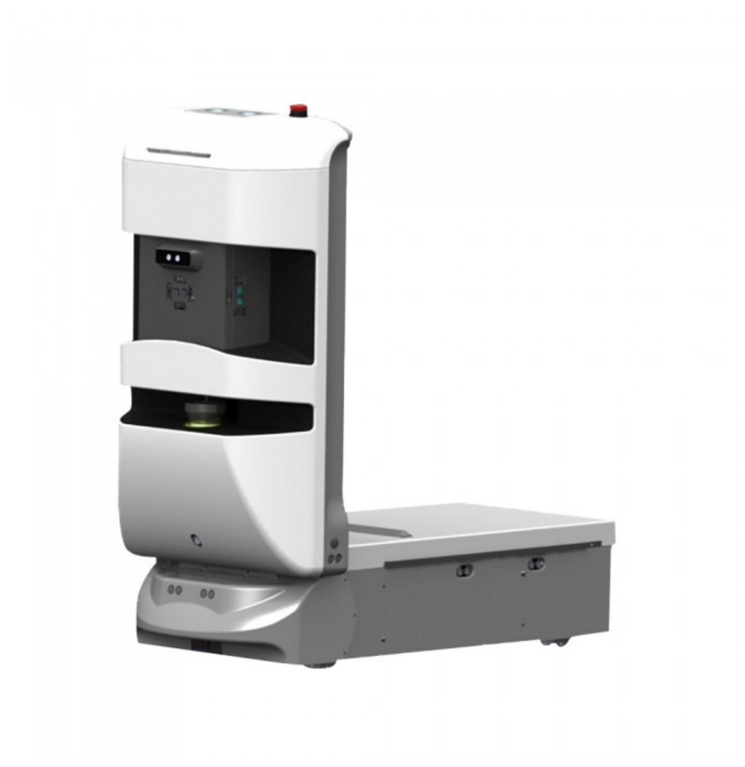


Figure 1.9: Aethon TUG T3

tion, Knightscope K3 has also a text to speech implementation. His peculiarity is that it has not a touchscreen interface but uses voice to create an interaction between the end user and the Knightscope Security Operations Center, a browser-based user interface[11].
The fifth and last product that has been analyzed was *AIRSTAR*, an intelligence autonomous robot produced by *LG*. The field of application, that is exactly indoor navigation in an airport environment, led Innotech Sys. to take a strong cue from this kind of product. AIRSTAR is a robot incorporated with various ICT technologies such as speech recognition and artificial intelligence, and is unanimously considered very innovative in terms of design and functions since it has been the worlds first commercially operating robot in this field.
The robot is able to guide passengers in both terminals of Incheon International Airport of Seoul, including assistance for departure and arrival halls, duty-free zones and baggage claim areas but it is also able to provide useful information in real-time like the congestion status at the departure halls. Travelers can obtain useful information about the airport communicating with AIRSTAR in different ways: by means of speech recognition algorithm, touching its touch-screen panels, or via barcode recognition through which the robot will inform them of their check-in counter. Moreover AIRSTAR is able to interact with passengers in four different languages: Korean, Chinese, Japanese and English [12]. Considering that the project Innotech Sys. was developing had a very similar mission AIRSTAR has been deeply ana-

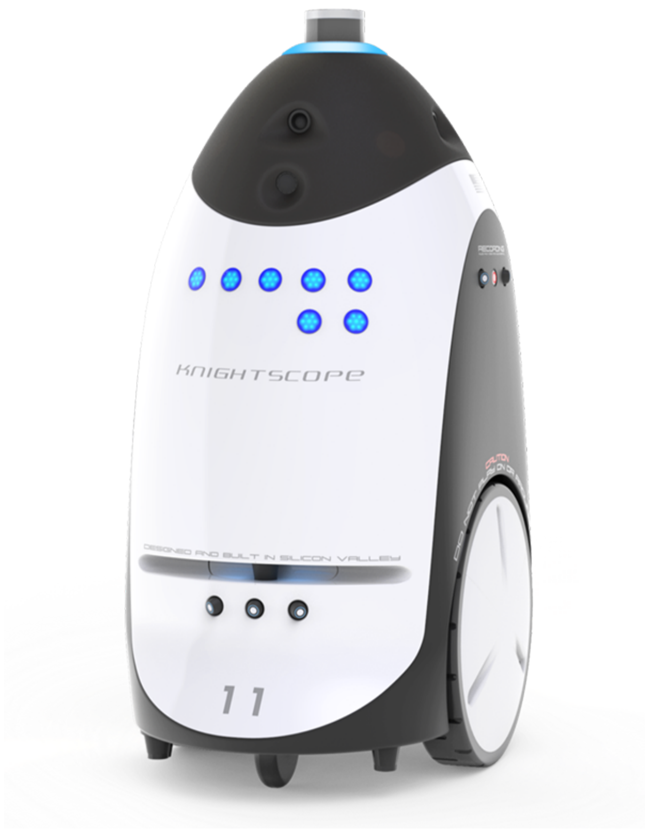


Figure 1.10: Knightscope K3

lyzed, especially regarding its human-robot interface. After having analyzed all the existing systems, a design strategy has been chosen on the basis of time and budget requirements.

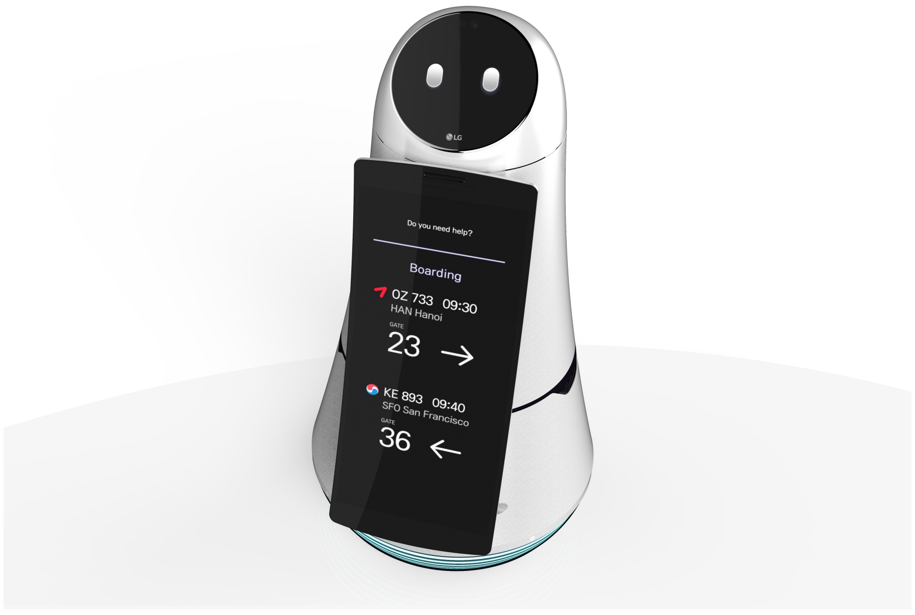


Figure 1.11: LG Airstar

Chapter 2

Interface Design

Summary

In this chapter the interface design is explained in detail, presenting first of all the most recent stage in the development of robotic user interfaces and then focusing on the material design and its main guidelines for creating an Android product. After this brief elucidation, the proposed solution to the problem is presented on the basis of the design requirements, focusing on the integrated development environment and trying to explain the structure of the code and the classes as they have been organized.

2.1 State of the art and design requirements

Nowadays, thanks to the extremely rapid progress in the field of artificial intelligence, simulation and understanding of verbal language, there are more robotic applications oriented towards human-machine interface (HMI). This latter term is understood to mean a set of hardware-software technologies that allows the user to interact with a machine for the performance of its assignments. In order to make this interaction as simple as possible, HMI has evolved into a Graphical User Interface (GUI), that is able to successfully communicate with the user minimizing the number of operations that he has to perform in order to impart a command. Over time touchscreen-based interfaces have become the standard for GUIs, replacing gradually the use of physical buttons and switches in every field, from industrial machines to mobile devices. It is possible to state that since Apple in 2007 has introduced the very first version of iOS, a new age of human-machine interface has been born. From that moment on, this type of interaction has become of daily use, leading more and more companies to adopt a similar type of solution for their human-machine interfaces, in order to trick people into the sensation of being using a well known device. According to prof. Akram Hossain in his publication about Human-Machine interfaces, HMI plays a significant role in creating a friendly visual environment between the user and the technology. It is considered to be the window to the automation control system [13]. The same solution has been adopted by robotic companies as well, which started to develop interfaces as similar as possible to the smartphone ones with the aim of providing to people a pleasant experience, based on a friendly and intuitive interaction. Bearing in mind the already discussed problem of a common negative feeling with regard to robots, this kind of case plays a role even more important cause it has the multiple mission of inspiring trust while being useful and beautifully designed. Even if the robot that has been designed is referred to as an autonomous system, it operates in response to user instructions and interacts with a human operator or with the the other hardware components communicating its results and its status. For this reason the design of a suitable Robotic User Interface has been required. In the first place the idea was about the possibility of creating an humanoid interface, with a screen that simulated a human face, so a case studies has been conducted. Unexpectedly, as highlighted by the study conducted by prof. Mori in his essay *The uncanny valley* [14], it is important that the appearance of the robot is not too human in order to not create false expectations into users; indeed, sometimes this can be counterproductive as shown in figure 2.1. As it possible to

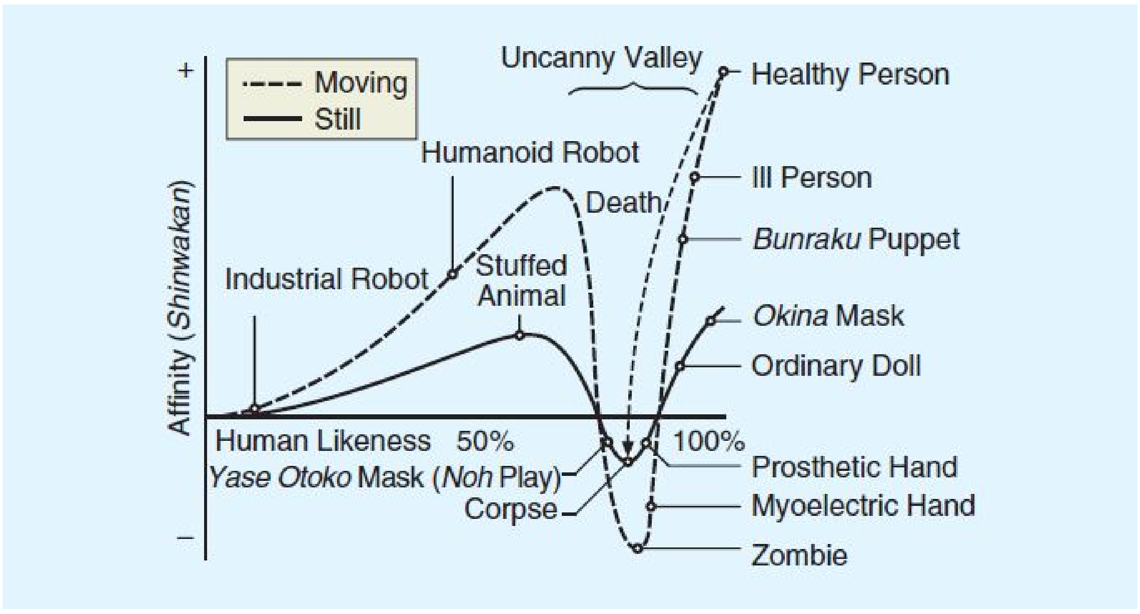


Figure 2.1: Uncanny Valley

notice, the peak is even amplified in case of mobile robots (dashed curve) creating in the user the sensation of being related to something it looks human but it's not, called *Zombie* in the figure, as a matter of fact. The humanoid aspect, therefore, must be suitably balanced based on how advanced his behavior is. Having in mind that the human-robot interaction had to be both screen-based and speech-dialogue based, a case study about most recent stages in the development of robotic user interfaces has been conducted.
As already mentioned, the marketed product that has been deeply analyzed was

AIRSTAR service robot by LG that had the very same objective of passenger guidance and sharing of useful information. Moreover this robot has indoor navigation in an airport environment as area of application, so it has been possible to take a leaf for as concerned the Robotic User Interface. LG developed a complete smart interface with the main goal of giving the impression of something friendly and intuitive, easy to use the first time. For this reason a interface similar to the one it can be found on a smartphone has been chosen, with the main menu based on some options including for example Language Select, Ticket Scan, Location info or Guide & Escort as shown in figures 2.2 and 2.3. It is worth to notice that every

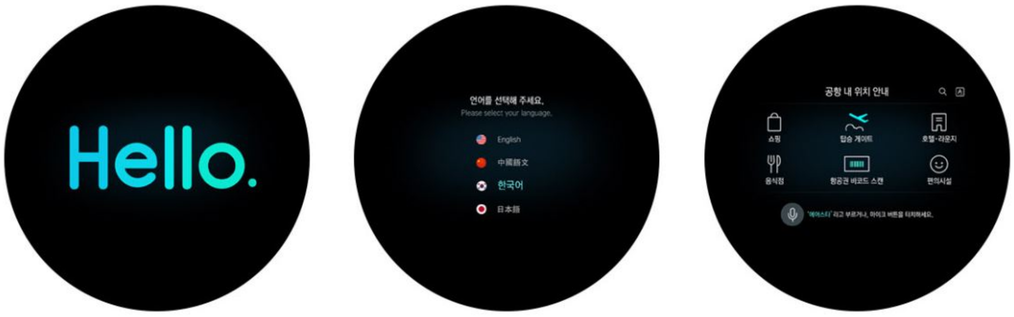


Figure 2.2: AirStar GUI Main Menu

option is well designed, and very intuitive, thanks as instance to a toolbar that let the user to easily come back to the homepage or thanks to the microphone button well highlighted through which easily establishing a communication with the robot. AIRSTAR is equipped with two screens, a small one positioned above that per-

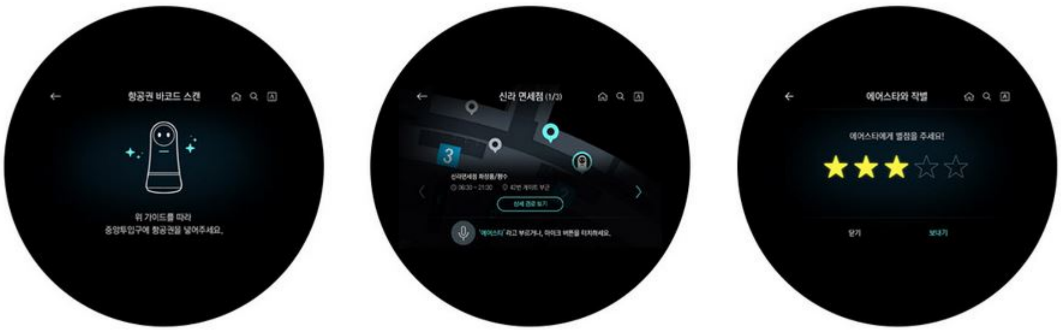


Figure 2.3: AirStar GUI Navigation Menu

forms the function of actual human-robot interface, by receiving command signals through the touchscreen panel, and then a big one positioned below that shows the information of status of the navigation or the path to follow (figures 2.4 and 2.5). After having analyzed the the state of the art in the development of a robotic user interface, a design strategy for the interface has been chosen. Many valid approaches

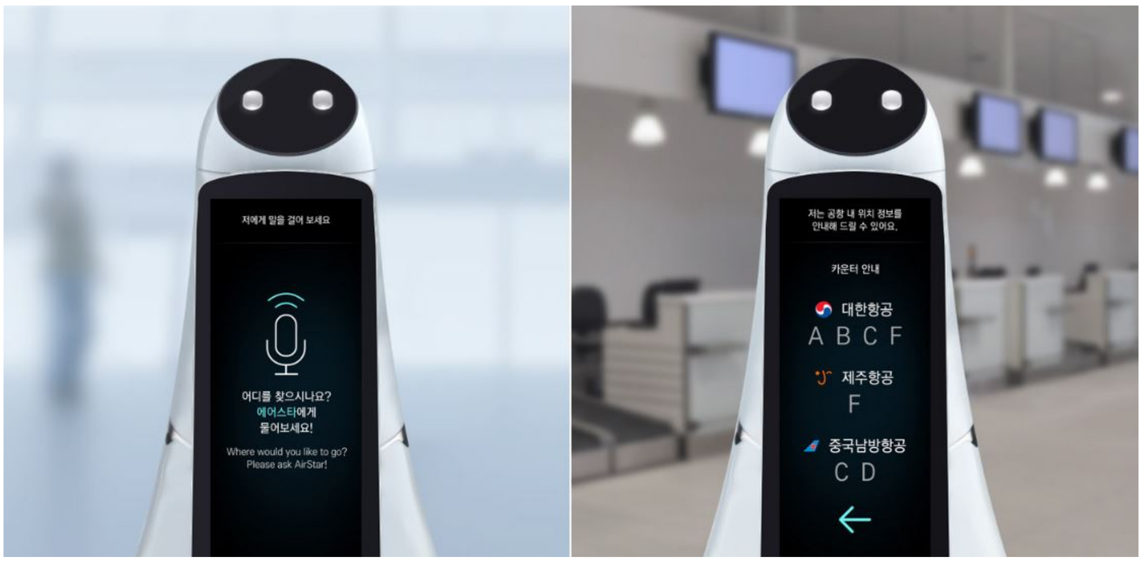


Figure 2.4: AirStar GUI Speech Dialogue Interface

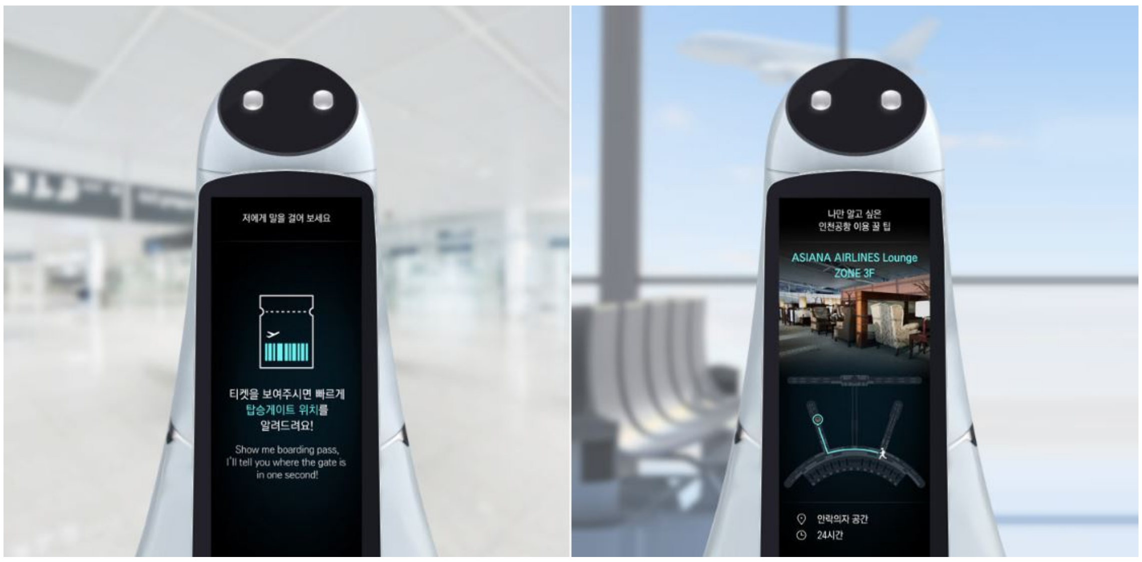


Figure 2.5: AirStar GUI Boarding Pass Scanning

have been taken into consideration, and choosing between them has been a trade-off between the required performance, allowable hardware integration time and budget. In general, the scheduling and performance needs of the User Interface are very different than those of the rest of the robot, so for this reason, it has been chosen to run the interface on a separate hardware device. An Android tablet has been selected because of its very affordable price, so there was not a big cost penalty to having a separate processor deploying the User Interface. Nevertheless, this kind of choice comes not without any drawbacks. The main problem linked to this kind of solution is the integration of another platform with respect to the robot main brain, that has its own software ecosystem. In fact while the entire SLAM and object detection algorithms are run on the Jetson TX2 and are coded in C++, the selected tablet supports Android language, that is to say Java language. Despite this drawback, it has been selected an Android based tablet for all the advantages that this kind of implementation brings with it, like the huge ecosystem of developers that has been very helpful during the implementation of the program or the excellent integration between the purely code level and the design of the graphical interface provided by the Integrated Development Environment *Android Studio*, as will be explained on the following pages.

2.2 What is Android?

Before going into the technical details of the Robotic User Interface implementation it is worth to briefly analyze the main characteristics of Android, the most widespread mobile Operating System present in the world. The very first Android commercial device has been launched by Google in 2008, and, from that moment, this platform started growing without pause till obtaining nowadays a percentage of the market equal to 75.27% [15]. This incredible result has been possible especially thanks to a huge ecosystem of developers, very productive and with a lot of future potential that has been created by providing right off the bat an Open Source software. Although Android is a mobile operating system, it is not only oriented to smartphone and tablets but also to smartwatch, wearable devices, televisions (Android TV) and even cars (Android auto) continuing the huge trend of economic growth of the Californian company. The system is based on a quite complex architecture built upon a Linux kernel, since it has been designed with the main mission being flexible and easily updatable.
As it is possible to see in figure 2.6, the architecture is divided in six basic components, very well explained by the official Android developers website [16]. On the bottom it is possible to find the Linux Kernel through which all the drivers, hardware components and memories are managed. It is worth to remember that the Linux Kernel is an example of monolithic kernel, that is to say a kernel where all the services share the same memory space, called User Address Space. This kind of implementation has the advantage of being faster in executing system calls or calls between operating systems components but the drawback of being less reliable because of the quantity of code running in kernel mode and the other drawback of being not so secure because malfunctions can easily propagate among the operating system components and corrupt the entire system. Just above it is possible to find the hardware abstraction layer (HAL) that consists of multiple libraries, through which hardware components are handled. Basically when an API makes a call to access device hardware, the Android system loads the library module for that hardware component creating basically a sort of interface. The above layer is filled by Android Runtime and Native C/C++ libraries. The former is used to compile the code minimizing memory footprint, while the latter, as the name suggests, is a set of native libraries that are at the root of the system and don't need to pass by the Android Runtime because they are already written in native code. Subsequently, there is the Java API Framework that includes all the set of classes, packages and interfaces used by developers for creating their apps. It is the most important layer since it handles the interface with lower layers avoiding the programmer to taking care of them. As well as official Java APIs, there are also unofficial, third-party APIs that provide additional functions.They can be downloaded from external websites, simplifying the life of the programmer, like as instance, GitHub libraries. The highest layer in the Android system architecture is taken by the System Apps, a set of default apps that can provide key capabilities that developers can access from their

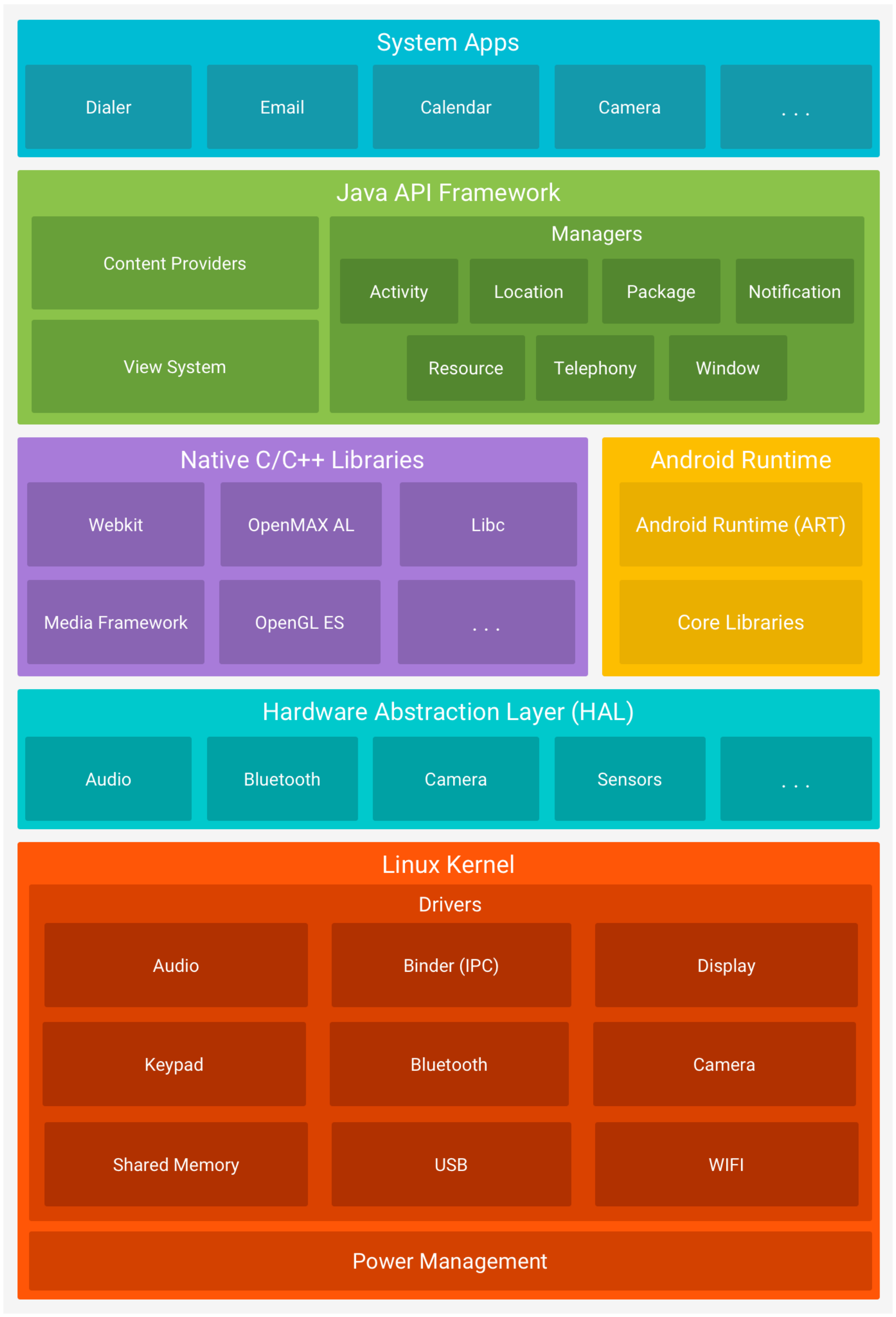


Figure 2.6: Android architecture

own app. As explained in the official Android Developers guide, these apps are not fixed and have no special status but can be easily substituted by other third-party apps. Even if most of these layers are not used by programmers, it is worth to know the internal architecture of the system in order to better understand how it works and trying to develop an app that handles properly the resources that it has at its disposal.

2.2.1 Android Studio IDE

The second area to analyze in order to start programming, after having understood the internal architecture of the whole Operating System, is the Integrating Development Environment (IDE) Android Studio by which every Android application is written and developed. It has been introduced by Google and IntelliJ in 2015, going to replace Eclipse IDE and it has been built specifically for Android development, in order to maximize flexibility and security. Android Studio can be considered as the official IDE for Android development, including inside it everything necessary for designing a complete project. Included in the download kit, there is the Software Development Kit (SDK), with all the Android libraries, and the infrastructure to download the Android emulators, used to test and debug the application, without needing a real device [17].
Each Android project is characterized by four components: Activity, Layout, Values and Drawables. The Activities are user interfaces on the basis of each Android project since they are the main Java classes in which all the actual Android code is implemented. An usual Android project contains more than one Activity, in order to facilitate readability and to optimise the app callings. In fact, when an app calls another this process can be done from a specific activity rather than from the whole app, minimizing the steps for performing a specific task. All the Java code written by the programmer is contained in the activities, that link the graphical user interface implemented in the Layouts with an actual action.
The Layouts are xml files, that contain the Android xml code through which the aesthetic part is implemented. Basically the layouts describe the graphic components that characterize the specific Activity and set their position inside the screen. There are different layouts through which it is possible to describe components but the most used are Linear Layout, Relative Layout and Constraint Layout. The first one, as its name suggests, aligns components horizontally or vertically setting the so called *Orientation* parameter.
The second one, that is the most used one, aligns components in a relative way with respect to each other using parameters like left-of, below, bottom, left, center. So, practically, each component is defined with a position relative to the previously defined components.
The third layout is considered as a sort of updated version of the Relative Layout. In fact, even if the Relative Layout is the most used layout for its simplicity, it has some drawbacks like, as instance, the big number of nested views in case of complex layouts, that reduces readability of the entire code. The aim of the Constraint Layout is precisely to help reducing the number of nested views anchoring any side of a view rather than having to place a whole view. Once the type of layout has been chosen and the components have been positioned, the connection between the current layout and the corresponding activity has to be made. In this way the com-

ponents just created correspond to a real action in the app like, as instance, the implementation of a specific action when a button is clicked.
The other two components that characterize an Android project are Values and Drawables. The former is a folder that contains all the information about themes, colors, styles and strings used in the development of the layout. Having all organized let the programmer to know exactly where to go when he wants to modify some graphical aspects of the app. The latter is still a folder but that contains all the images, icons or backgrounds that have been used during the development of the layout. It is worth to say that all these folders are automatically built by the compiler as soon as a new project is created and are selected and categorized on the basis of the quality of the image. As previously mentioned, the most important component into an Android project is the Activity, that has to be written by the programmer having in mind the main principles of every Android app, that is to say the security and the optimization of the resources to reduce energy and memory consumption. These principles are pretty intuitive to be understood, since every user looks for a device that is fluent, powerful and secure, and these characteristics are not only reached with a powerful hardware, but also with an optimized software able to manage all the real-time resources. In order to guarantee these principles the activity lifecycle has to be analyzed in detail.
As it is possible to see in figure 2.7 taken from the official Android Developers website, there are a lot of methods that are called after the activity is first created. The very first method that is called is the *onCreate()*, in which all the initializations have to be done. This method also contains all the associations between the components created in the Layout, like buttons, imageViews, textViews or listView and their java variables that will be used to perform a specific function. This method is followed by the *onStart()* that is called every time the activity becomes visible to the user. This method completes very quickly and as soon as this callback finishes, the activity enters the Resumed state, and the system invokes the *onResume()*, called every time the activity starts interacting with the user. At this point the activity is at the top of the stack, with user input going to it. When an interrupt occurs, the activity enters the Paused state, and the system invokes the *onPause()*. In this particular situation, the activity is going into the background, but it has not been killed yet. The last two methods to be called are *onStop()* and *onDestroy()*. The former is called when the activity is no longer visible to the user because is going to terminate. The latter is called only and only if the activity has to be destroyed. This can happen because it has finished its operations or because of memory or energy optimization[18]. It worth to notice that the *onStop()* method should be never called in low memory situations where the system does not have enough memory to keep the process running after its *onPause()* method is called.
The last method that has to be analyzed is *onRestart()* that is separated from the rest of the flow, as it is possible to see in figure 2.7. This method is called after the activity has been stopped by means of the method *onStop()* like, as instance, if the user presses the home button in the application. When this happens *onPause()*

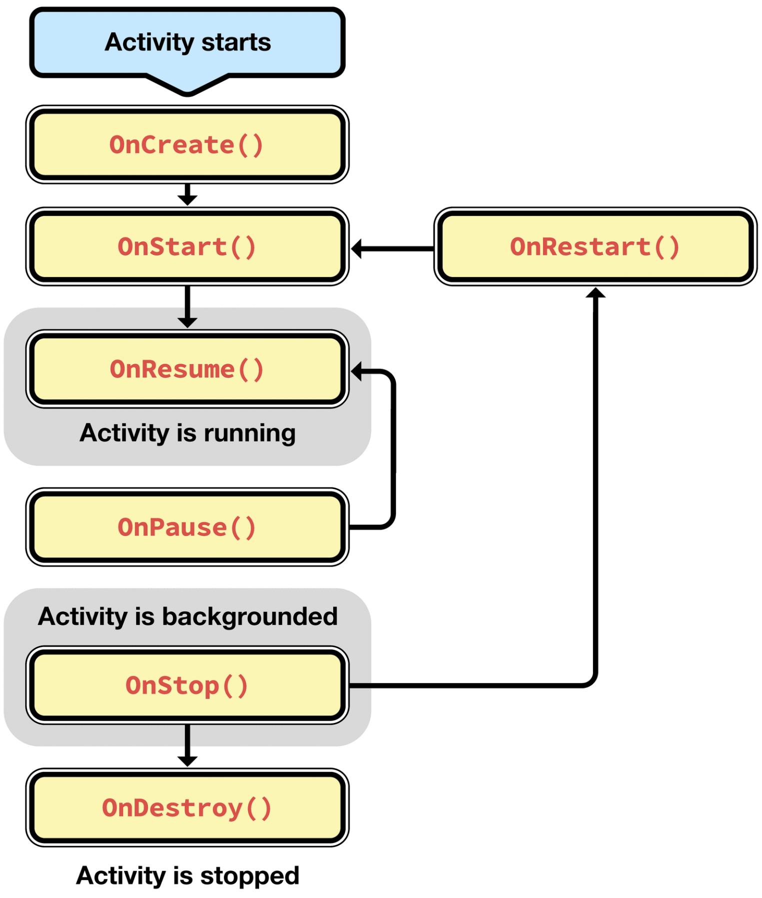
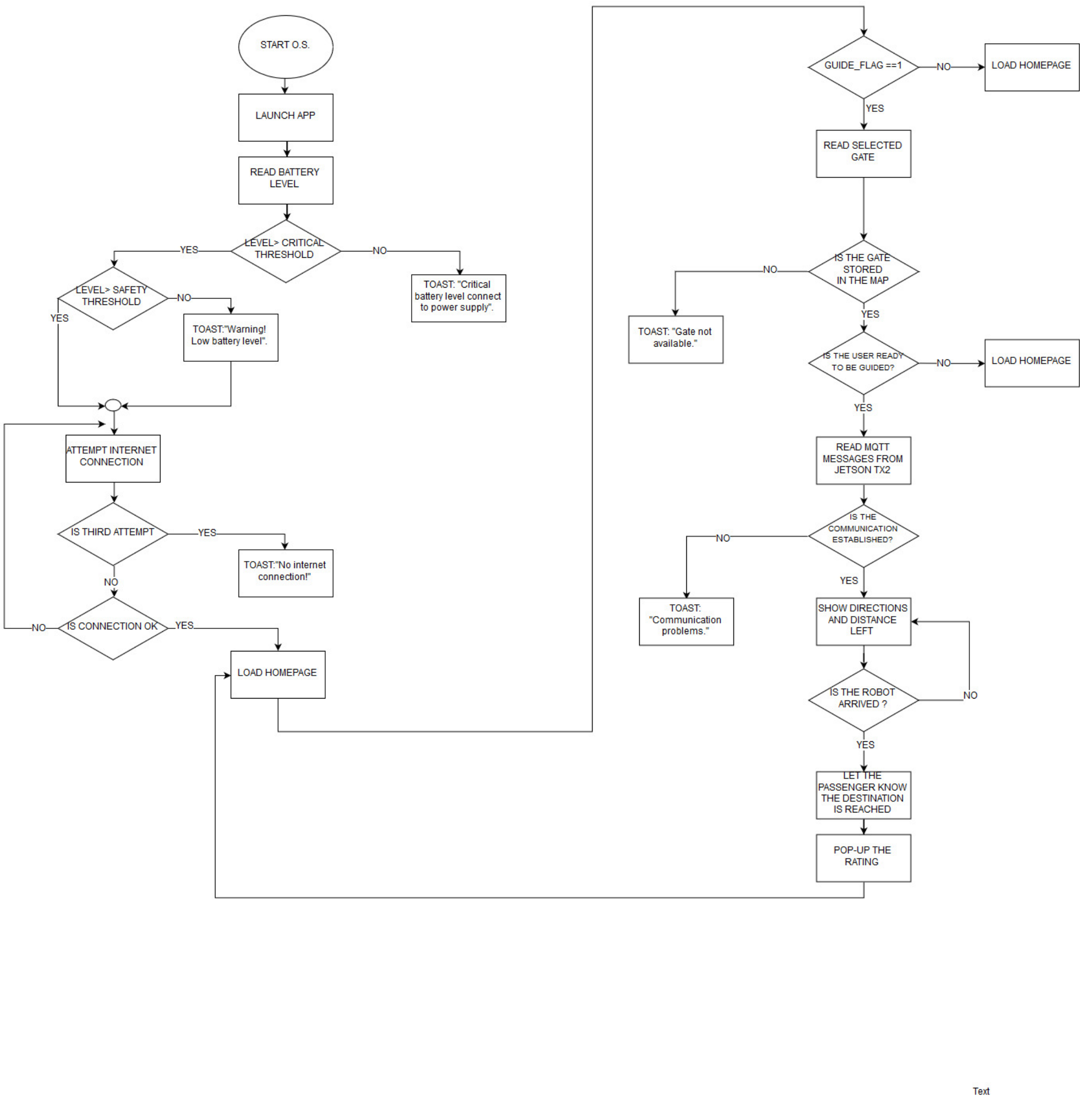


Figure 2.7: Activity Lifecycle

and then *onStop()* methods are called, and the Activity is moved to the background without being destroyed. In other words, it can be seen as the method that let the activity to be restored after having been put in background, without destroying it, and so reducing time to access to that particular activity. As already explained this method can be called only if there are no problems of memory or energy optimization otherwise the *onDestroy()* method is called. It is worth to notice that, during the development of any Andorid project, the only method that has to be written by the programmer is the *onCreate()* because all the other methods are handled and optimized automatically by the compiler unless the programmer wants to perform a specific action when one of these other methods is called.

2.3 Robotic User Interface Flow Chart

Once all the theory about how to develop properly an Android project has been analyzed, it is possible to understand in detail how the specific project of the Robotic User Interface (RUI) has been structured and implemented. In order to better understand how the interface has been implemented a flow chart has been drawn so that each action performed by the user is represented as a step in a graphical sequence, for sure more intuitive and pretty straightforward. Among all the possible operative situations, the one in which the user needs to be guided to a particular gate has been analyzed.



START O.S.
LAUNCH APP
READ BATTERY LEVEL
LEVEL> CRITICAL THRESHOLD
YES
NO
LEVEL> SAFETY THRESHOLD
NO
YES
TOAST: "Critical battery level connect to power supply".
TOAST:"Warning! Low battery level".
ATTEMPT INTERNET CONNECTION
IS THIRD ATTEMPT
YES
NO
TOAST:"No internet connection!"
IS CONNECTION OK
NO
YES
LOAD HOMEPAGE
GUIDE_FLAG ==1
NO
LOAD HOMEPAGE
YES
READ SELECTED GATE
IS THE GATE STORED IN THE MAP
NO
TOAST: "Gate not available."
YES
IS THE USER READY TO BE GUIDED?
NO
LOAD HOMEPAGE
YES
READ MQTT MESSAGES FROM JETSON TX2
IS THE COMMUNICATION ESTABLISHED?
NO
TOAST: "Communication problems."
YES
SHOW DIRECTIONS AND DISTANCE LEFT
IS THE ROBOT ARRIVED ?
NO
YES
LET THE PASSENGER KNOW THE DESTINATION IS REACHED
POP-UP THE RATING
Text

Figure 2.8: Flow Chart

As it is possible to see in figure 2.8, as soon as the Robot is turned on, the user interface is activated and it is programmed to read the battery level of the robot. An actual communication between the tablet and the batteries has not been implemented because of the useless increase of the cost due to additional wiring systems and interfaces. For these reasons it has been chosen to read the battery level from the F28069-LAUNCHXL evaluation board, that sends the value to the Jetson TX2 via CAN bus that finally sends the value to the tablet via MQTT protocol. Once the tablet has the battery level, it compares it with a critical threshold and with a safety threshold in order to understand if the battery level is not sufficient to accomplish any kind of task or if it is high enough. As it is possible to see in case the battery level is under the minimum threshold a toast is displayed on the screen that informs the user or the maintenance worker about what is going on with the batteries. Supposing that the battery level is sufficiently high, the second thing that the algorithm checks is the internet connection,in particular it makes up to three attempts after which another toast is displayed informing the user about the problem. Finally, if the connection has been established, the homepage is loaded that is to say the corresponding Main Activity of the Android Studio project. At this point, as soon as the user taps the gate button or asks for a guide by means of the speech recognition algorithm the *Gate activity* is called and the gate selected by the user is stored in a variable and sent to the Jetson TX2 board through MQTT protocol. At this stage the interface is paused waiting for a response by the Jetson TX2 board: if the gate selected by the user was already stored and mapped, the Jetson board informs the Robotic User Interface that is ready to start the guidance otherwise if the gate is not recognized inside the map, the user is immediately informed by means of a toast on the tablet screen. If both the robot and the user are ready to go, the *MQTT activity* is called and the communication real-time about the status of the navigation is established. In particular the information about the distance is updated on the screen so that the user can know how far he has to walk as well as the information about the direction of the navigation by means of blinking arrows displayed on the screen. As soon as the robot is arrived to the established destination, the last message is sent from the Jetson board to the tablet. This message triggers another object of the Media Player class that starts a voice that informs the user that the destination has been reached, and right after, an alert view with a rating survey pops up so that the user can rate its experience.

2.4 Material Design

As previously discussed the design of a graphical user interface plays a fundamental role in the development of robot that aims to inspire trust and appreciation. Material Design is a visual language that has been introduced by Google in 2014 just because the company wanted to create a way to merge the classic principles of good design with the innovation of technology and science. It is characterized by a set of guidelines that have to be followed in order to obtain high-quality output, giving users sensation of dealing with components that behave like real-world objects. The main idea of Material Design is to apply basic, natural laws from the physical world, principally concerning lighting and motion with the purpose of minimizing mental effort and ambiguity. That is what Ian G. Clifton says on design in his book Android User Interface Design:

> *"Design has many purposes, but two of the most important are usability and visual appeal. You want brand-new users to be able to jump into your app and get started without any effort because mobile users are more impatient than users of nearly any other platform. Users need to know exactly what they can interact with, and they need to be able to do so in a hurry while distracted"* [19].

Bearing in mind all these preliminary notions, a Graphical User Interface has been developed starting from a 3D layered structure. The most internal layer is represented by the background image upon which a card layout has been implemented. The chosen background image was an image of San Diego Airport that automatically changes day to night view depending on the time. The background has been set with an opacity of 40% so that card layer built upon it is sufficiently noticeable and does not take the attention away from user. Cards are referred to be surfaces that display content and actions on a single topic (figure 2.9). In the specific implementation, the following items have been implemented in Card Layout:

- Floating Action Button
- Toolbar
- Image Buttons
- Bottom Navigation Bar

In the next pages each of these components will be explained in detail, focusing on how they have been implemented in the Robotic User Interface design.
As reported on the official Android developers forum, a Floating Action Button (FAB) is a button shaped like a circled icon floating that triggers the primary action in the app's User Interface. Thanks to the fact that it expands and collapses when clicked, FAB contains a lot of useful information but at the same time it takes little room on the screen, reducing the sensation of misplacement in the user. FAB

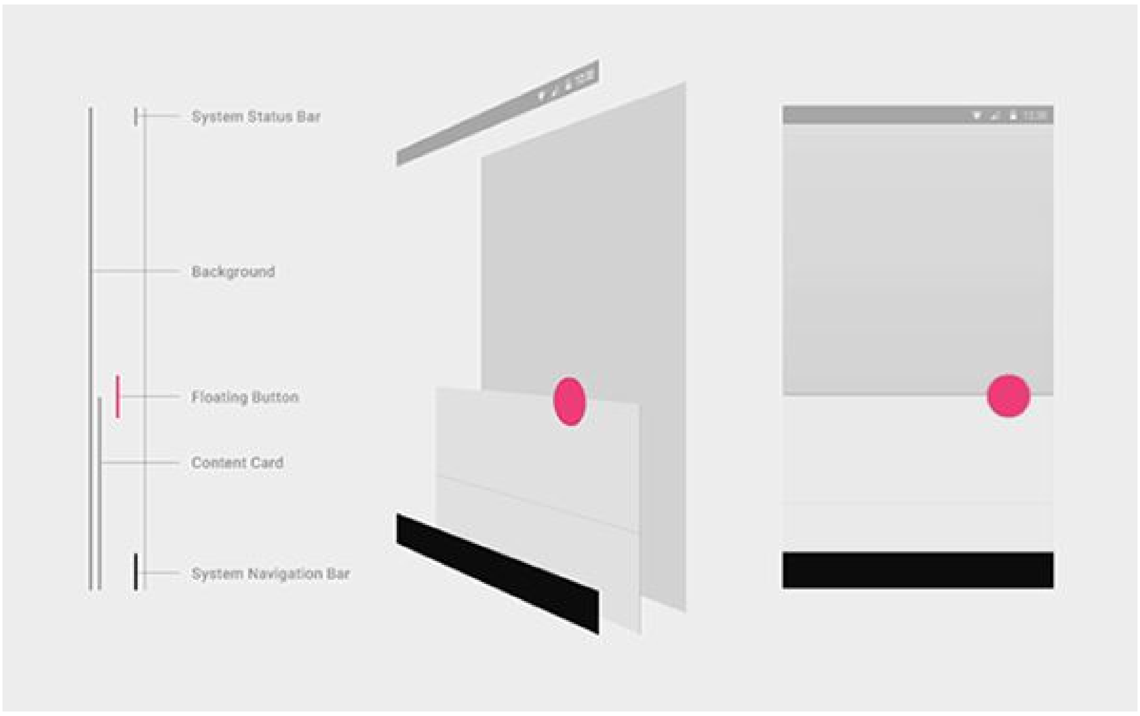
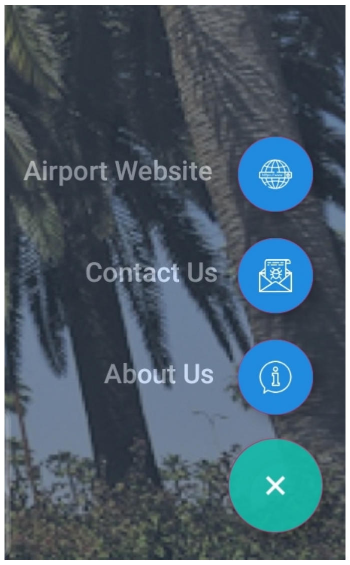
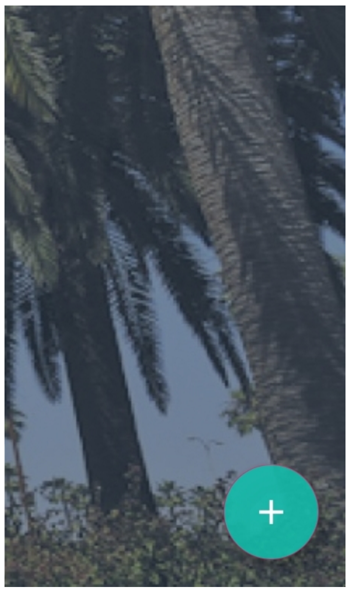


Figure 2.9: Example of layered structure

represents a natural cue for helping users when they faced with unfamiliar screen, cause it has been proved that in case of confusion users have a tendency to rely on it to navigate. According to Material Design guidelines FABs have to be colorful, raised and grid-breaking in order to draw attention to the user and they should follow the principles of being Primary, Constructive and Contextual [20]. Bearing in mind all these guidelines, a Floating Action Button has been implemented, with a clickable main button always visible and three secondary buttons visible only when the main button is clicked.
As it is possible to see in figure 2.10a, as soon as the primary button is clicked, three buttons pop up with their labels, providing to the user the possibility to navigate on the San Diego Airport website, or contact the company by e-mail, or have more information about the company Innotech-Sys directly on the company website. In order to make everything more appealing, an animation has been implemented any time the FAB expands or collapses.
The second item that has been implemented in Material Design was the Toolbar. The toolbar bar (formerly known as action bar) can be freely positioned in the layout file and can host icons or a back button that can be very useful for navigating through the activities of the app. The toolbar bar substituted the action bar from Android API 21 and above with the main mission of improving customization like, as instance, position on the screen, color, opacity, text size, icons and many other items[20]. As it is shown in figure 2.11, the toolbar bar that has been implemented in the homepage of the interface is characterized by four icons. The rounded flag



(a) FAB expanded



(b) FAB collapsed

Figure 2.10: FAB structure

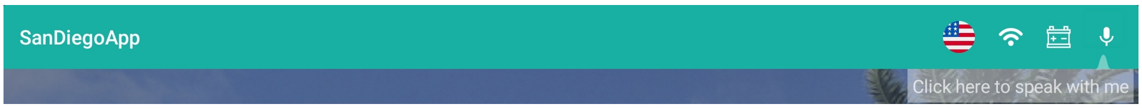


Figure 2.11: Toolbar Bar

has been used to represent the selected language. When clicked, this icon shows a pop-up window through which it is possible to change both the system language and the speech recognition language (figure 2.12). As will be explained in chapter 3, five different languages have been implemented, including English, Spanish, French, Italian and Japanese. Once the user clicks on one of the available languages, the interface reboots changing all the text labels, the corresponding flag icon and the default voice of the speech recognition algorithm. Next to the flag button, there are other three buttons, the Wi-Fi button that let the user to access directly to internet settings in case there are connection issues, the battery button that shows on the screen the battery level of the robot communicating with the internal boards and finally the microphone button that enables the speech recognition algorithm. As it possible to see in figure 2.11 at top right corner, a ToolTip has been implemented, that is to say a message which appears to help users to better understand the functionality of that specific button. This solution has been adopted after that in several tests has been encountered by the users a certain difficulty in making use of the speech recognition because the button was not sufficiently highlighted.
The third item that has been implemented was the set of image buttons. Buttons allow users to take actions, and make choices, with a single tap and, according to Material Design guidelines, they have to follow the three principles of being

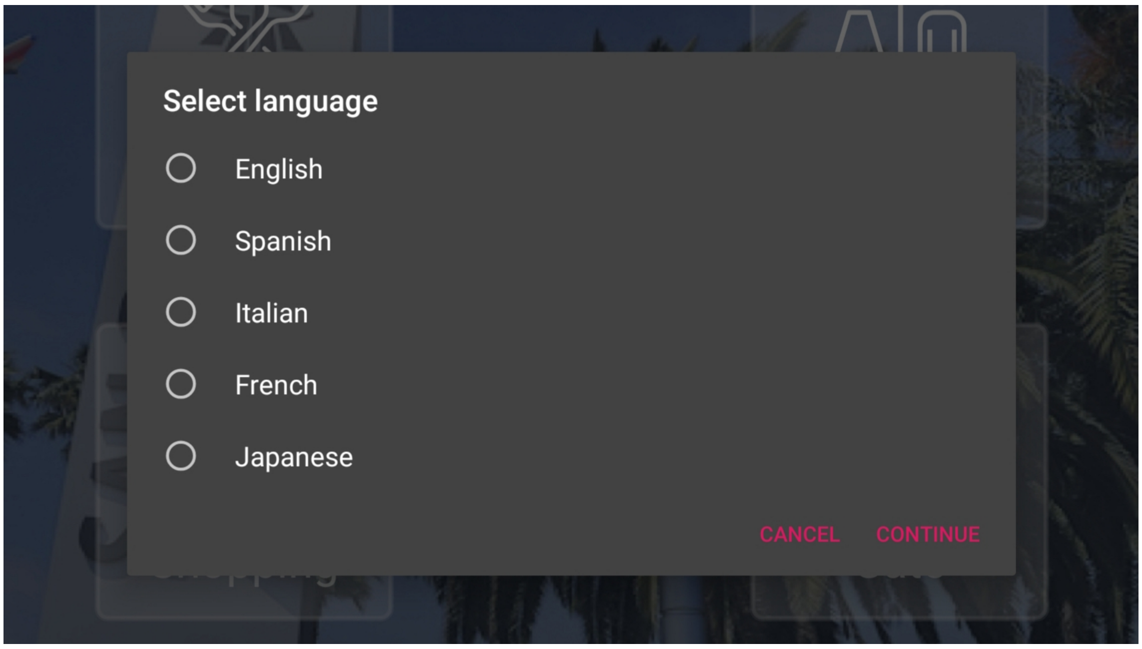


Figure 2.12: Language Selection

Identifiable, Findable and Clear[20]. In order to create buttons that were as intuitive as possible, image buttons have been selected that combine an illustration and a text label to maximize readability. So, a minimal icon pack has been chosen followed by a text label that explains the function of each specific button. Moreover, as regards dimensions, they have been designed to be as visible as possible, as well as easily clickable. For these reasons each of them has been included into a square frame of side 200dp with an opacity of 22% and round edges for improving the visual appearance.
As it is possible to see in figure 2.13, four buttons have been implemented: food, restroom, shopping and gate. Food button, once clicked, displays a second activity with two fragments linked to each other(figure 2.14). The first fragment shows a list of all the restaurants and bars available in terminal 1 of San Diego Airport, and has been implemented by means of an expandable list view. When the item representing a specific restaurant is collapsed the only things that are shown are the restaurant name and logo. As soon as the user clicks on that specific item, it expands itself showing more information like opening hours, location, guide button to be guided to the requested restaurant and a map that highlights the item that has been clicked. A similar structure has been adopted for shopping and restroom button, but the implementation of these button has been set aside because the airport security department did not provide access to airport database including maps of shops and restroom or flight information. As regards gate button, it has been implemented in a different way. When it is clicked, it shows a second activity where asks to the user which gate he wants to reach and after that it triggers the communication with

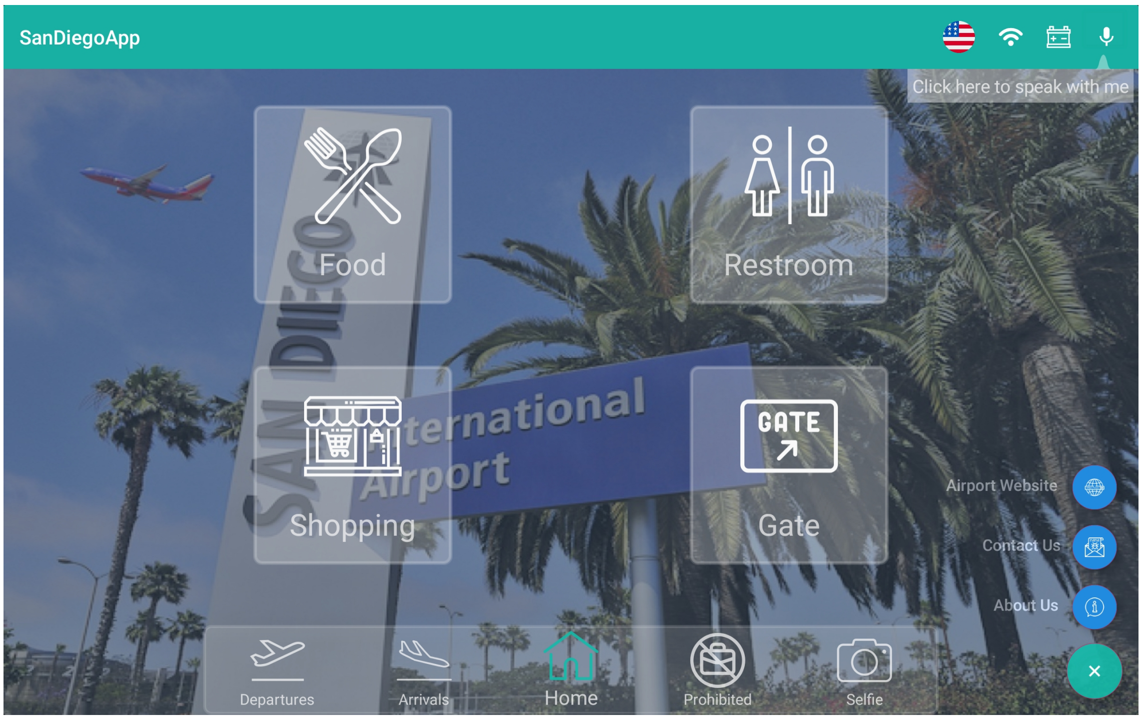
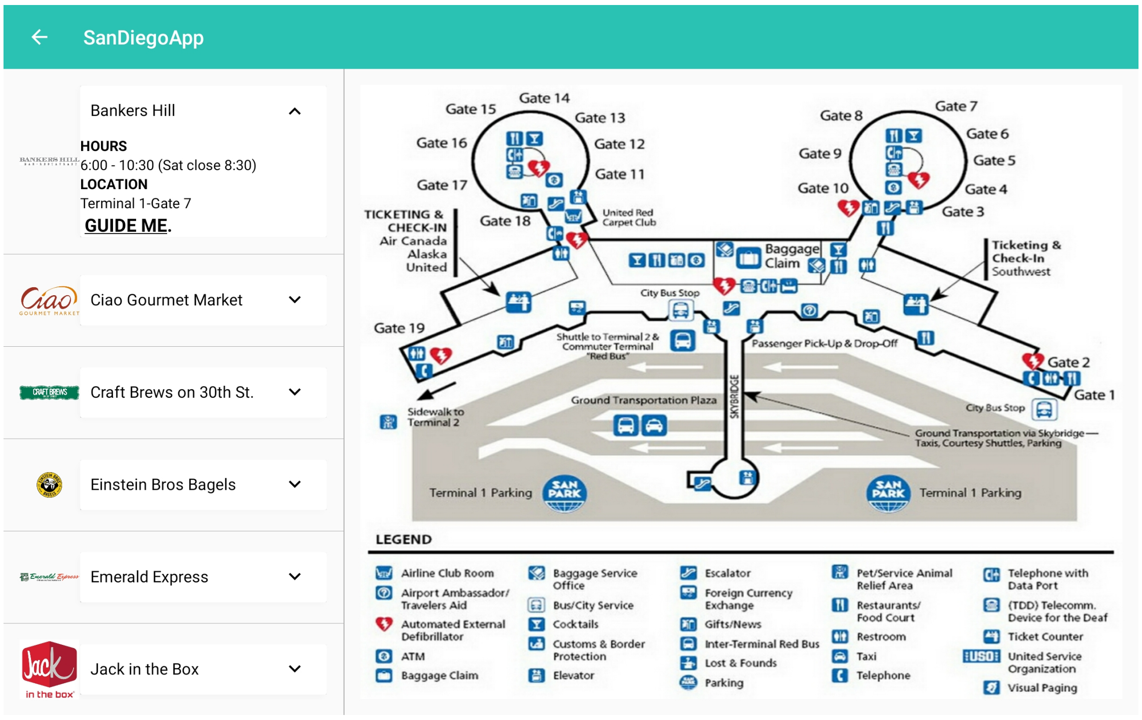


Figure 2.13: Button implementation



Bankers Hill

HOURS
6:00 - 10:30 (Sat close 8:30)
LOCATION
Terminal 1-Gate 7
GUIDE ME.

Ciao Gourmet Market

Craft Brews on 30th St.

Einstein Bros Bagels

Emerald Express

Jack in the Box

Figure 2.14: Food Map

the NVIDIA Jetson TX2 board for the navigation. Once the communication has been established and the user confirms its wish to be guided to the clicked gate, a new activity is created where the information about the navigation are displayed. In particular, as it is possible to see in figure 4.6 information like distance left and direction arrows are displayed as well as an emergency stop button that undoes the navigation.
The last item that has been implemented was the Bottom Navigation Bar. It is positioned at the bottom of the screen and allows to shift from one activity to another or from one fragment to another of the same activity. Each option is represented by an icon and a text label and it is highlighted as soon as clicked. As described by Material Design guidelines a Bottom Navigation Bar has to follow the three principle of being Ergonomic that is say it has to be easy to reach on a handheld mobile device, Consistent that is that it always appears at the bottom of every screen and Related that is to say that every destination has to have equal importance [20]. As it possible to see on figure 2.15, a navigation bar with five destinations has been

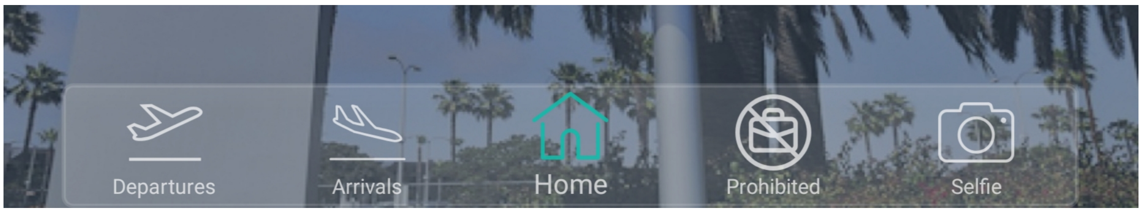


Figure 2.15: Bottom Navigation Bar

designed. Departures and Arrivals, when tapped, open a new activity with a Search View. At this point the user should type the requested destination or arrival and just the information related to that specific airport should be shown. Basically this choice has been made because it has been reported on the passengers interviews that the majority found the flight information screens very confusing and complicated so it has been necessary to think about something to come up with this inconvenience. As already explained San Diego Airport did not provide the access to the database but a formatted table has been created in advance so that as soon as the airport provides the access, the interface is already set to show the information. Prohibited button has been implemented to help passengers to check which kind of items they are allowed to bring on board. When the user taps on that button, a new activity is opened with a Web View that shows the website of the Transportation Security Administration directly inside the interface application, without an external usage of a browser. Selfie button has been implemented as a way to make easy the interaction with the robot. In fact, when this button is clicked, a smile face pops up letting the passenger take a selfie with the robot and share on social media. Starting from each of these destination it is always possible to come back to the homepage thanks to an intuitive implementation of the Bottom Navigation Bar that is present in each one of the different activities.

2.5 Code structure

The core of all the RUI is represented by the *Main Activity* so it is logic to analyze it first of all. The *onCreate()* has been the first method to be developed. This method contains all the connections with the various buttons, toolbars and all the other components created in the *activity main.xml* layout file. The various components are handled in a different way, so the analysis can start from the most simple management, the one that concerns the four macro buttons for Food,Shopping,Restroom and Gate. Basically as soon as one of them is clicked, the method *setOnClickListener()* associated to that specific button is called and, inside it, the creation of a new activity is triggered. This new activity needs the information about the language setting in order to create labels and to set the speech recognition algorithm accordingly. This information is sent to the new activity by means of an Intent.
As regards the Floating Action Buttons they are handled in the same way as the macro Buttons with the only difference that their openings and closings are done by means of an animation. The control on their status has been done using a simple status flag that toggles when a transition occurs. A different implementation has been adopted for the management of the toolbar and its items. First of all when one of these items is tapped the *onCreateOptionsMenu()* method is called outside the *onCreate()* method and a switch case is made in order to understand which item has been clicked.
In case the clicked item is the language selector, the method *selectLanguage()* is called and an Alert Dialog pops up on the screen allowing the user to select the language. Once the language has been selected, this information is stored in a String and is sent by means of an intent to all the other activities that need this information. In case the clicked item is the Wi-Fi icon, the Wi-Fi setting default activity is called, while, in case is the battery icon ia the one being clicked, the MQTT protocol is activated and the percentage of battery is read by the Jetson TX2 board that, in turn, reads the CAN message sent by the F28069-LAUNCHXL evaluation board. When the microphone icon is clicked, the *onOptionsItemSelected()* method is called and the default voice starts by means of an object of the class Media Player. Before doing that, the default voice language is selected by means of a switch case that selects the string already stored with this information. If the user interacts with the Robotic User Interface, the *onActivityResult()* method is called and the speech is analyzed performing actions depending on the user request. This analysis is done by means of Google Recognizer within the library *android.speech.RecognizerIntent* as will be deeply analyzed in Chapter 3.
After having explained all the functions and the operations of the *onCreate()* method, the *onRestart()* method has been developed. Even if, as already discussed, this method is entirely managed by the compiler it has been modified to show the tool tip close to the microphone button. Basically in this way, every time the homepage is restarted the tool tip is shown helping new users to find the microphone more easily. Besides the Main Activity all the other secondary activities have been im-

plemented for different purposes like the communication between tablet and Jetson TX2 board (*MQTT Activity*) or the interaction with the Airport Server and the developing of a table where parse a JSON file (*Arrival/Departure Activity*). All these activities develop different actions that will be analyzed in Chapter 4, but they are characterized by the very same code structure just studied, so that they can be easily understood having in mind all the notions discussed in this section.

Chapter 3

Speech Recognition

Summary

This chapter describes the case study that has been conducted about the most recent stages in the development of speech recognition algorithms. The adopted solution will be then explained focusing on the theoretical aspects behind its implementation explaining the reasons that led to that choice on the basis of time and budget resources as well as to consistency with the Android platform and design requirements.

3.1 State of the art and design requirements

After having developed the graphical part of the Robotic User Interface and having submitted the app to the end users, it has been figured out the importance of implementing a speech recognition algorithm to minimize the number of steps required for running a command and creating a more natural and intuitive Human-Robot interaction. Indeed, according to the Institute of Electrical and Electronics Engineers (IEEE), Speech recognition, also known as speech-to-text or automatic speech recognition (ASR), allows the machine to turn the speech signal into text or commands through the process of identification and understanding. Its ultimate goal is to achieve natural language communication between man and machine[21]. The simpler is the communication, the more effective it is and for this reason, as mentioned in the paper written by Zhong, Raman, Burkhardt, Biadsy and Bigha, more and more devices implement the speech recognition algorithm through a touch-screen. The advantage of voice commands over multi-touch is that it does not require the user to click physically on a portion of the screen and so it reduces time necessary to complete the specific task [22].
Although speech recognition has been studied for many years, nowadays has been widely adopted in many contexts like automatic customer service hotlines or virtual assistants like Siri, Google Assistant or Amazon Alexa [22]. The reasons about this sudden expansion have to be found in the technological progress that has increased

the quality of results like accuracy, speed and, in particular, capability of recognizing different speakers on the basis of their accent, dialect or physiology, thus establishing a smooth and intuitive Human-Robot interaction. The state of the art in this field, is represented by the most popular voice assistant, Siri, adopted in Apple ecosystem. Siri is a digital vocal assistant able to understand and process consequently questions posed in natural language by the end user. Present on every iOS device starting from iPhone 4S, it has been introduced on Apple Watch, macOs, iPod Touch and even HomeKit improving year by year further and furth,er functionalities related with weather, alarm and timer, calls and FaceTime, calendar, sport news and much more. Although the technologies behind the functioning of Siri are quite complex, it is possible to split the operating mechanism in four phases [23]:

1. Voice Recognition
2. Send everything to Apple servers in the cloud
3. Understand the meaning
4. Transform the meaning into actionable instructions

During the first phase, the device activates the microphone, collects the analog voice signal and converts it into a binary file that is then sent to the server. The generated audio file is characterized by different features like language, accent and, in particular, environmental noise that have to be filtered and analyzed. All these processes are computationally heavy for a mobile device, so it has been chosen to sent the raw file to the servers that have access to much more powerful devices and that can process it in a limited amount of time. This is what happens during the second phase of the operating mechanism in which it is pretty clear the need of an internet connection in order to let this technology work without causing a fast draining of the battery level as well as all the limited hardware resources of the mobile device [23]. After having filtered and processed the audio file, the algorithm has to understand what the user is trying to communicate. During this phase, called *natural language processing*, the algorithm analyzes the syntactical structure of the audio source and selects the flowchart branch that better fits with the processed file. Last but not least, there is the process of converting the request that has been processed into an operative action. Although it could seem pretty straightforward, this is not an easy task because it can be likely to occur that the text request coming from the user has been correctly processed and understood but the consequent action is not consistent with what has been understood by the algorithm. That is the reason why the algorithm does not process single words but tries to understand the context[23]. Another fundamental feature that has promoted exponentially the usage of Siri,as explained in a scientific paper published in *Apple Machine Learning Journal*, is the implementation of an hand-free access simply pronouncing the *"Hey, Siri!"* watchword, without the need of pressing the home button. As shown in figure 3.1, the first step is to detect the audio source file and analyze it in the slightest possible time.

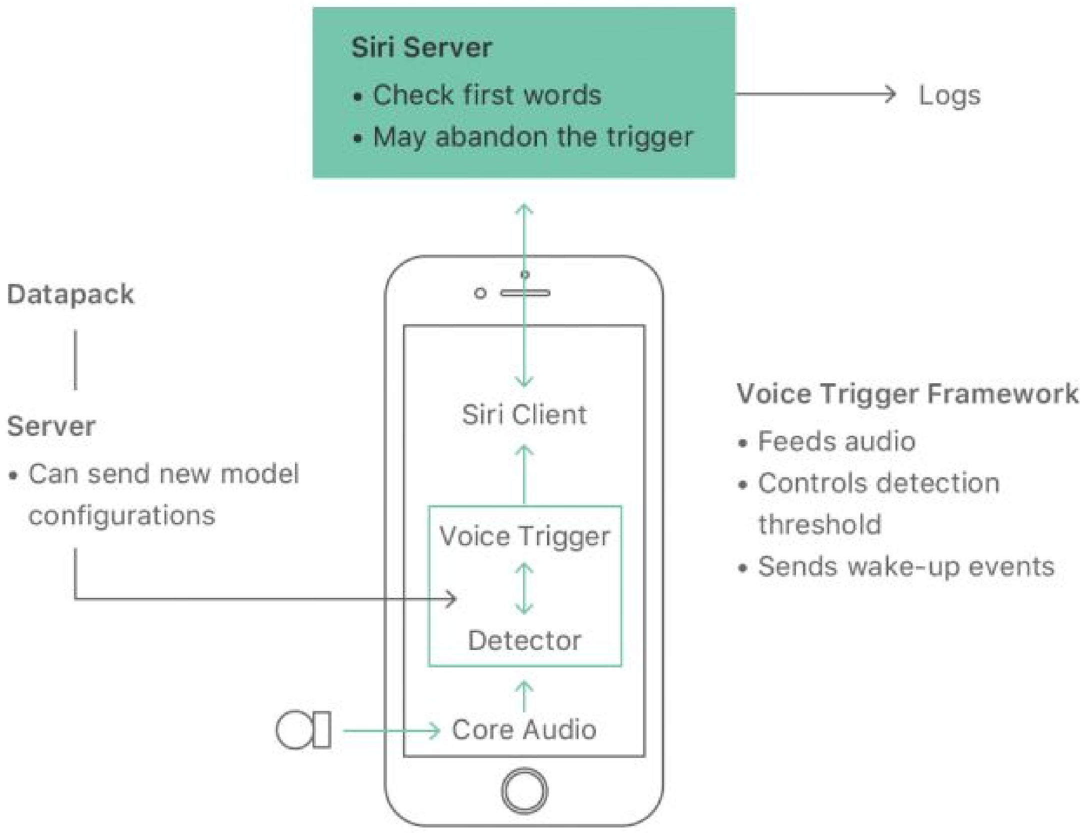


Figure 3.1: Hey Siri flow on Iphone[1]

This is done by converting the voice into a stream of waveform samples, with a rate of 16000 waves per second. About twenty of these frames at a time (0.2 seconds of audio) are transmitted to the neural network that converts these acoustic models into a set of vocal classes: those used in the phrase *"Hey Siri"*, plus the subsequent silence and any other sounds, for a total of 20 classes. In order to avoid false voice triggers, the Cupertino company decided to insert two thresholds, a primary one and a secondary one that normally does not activate Siri. If the score exceeds the lower threshold but not the upper one then is likely that the user is calling the *"Hey Siri"* event. This algorithm provides for the cooperation among hardware, software and internet services and requires the processor to listen for the trigger phase all the time, causing a huge resources draining. That is the reason why a low-power Always On Processor has been embedded in the Motion Coprocessor improving accuracy and performances of the entire process[1].
After having conducted a case study about most recent stages in the development of speech recognition for mobile devices, it has been done the same for Human-Robot interaction field, focusing in particular on the implementation of Voice Command Detection adopted in NVIDIA Isaac SDK. The NVIDIA Isaac Software Development

Kit (SDK) is an open source toolbox made available to all the developers that want to grapple with autonomous robots. It includes not only the algorithm for voice command detection but also libraries for speeding up the development of Artificial Intelligence features like perception and navigation. Based on C++ and Python, it supports only Ubuntu 18.04 LTS and requires the NVIDIA Jetson Nano board that is a cheaper and less powerful version of a Jetson TX2 board already mentioned in the previous chapters.
As regards the implementation of the Voice Command Detection feature, it implements a simple and lightweight Human-Robot interaction based on a limited set of short commands that can be understandable by the Robot and that can be converted into actionable instructions. This kind of algorithm does not require considerable resources because it is characterized by a small vocabulary so it can be deployed on a Jetson nano board, reducing costs for the hardware. It is based on 3 codelets: Voice Command Feature Extraction, Tensorflow Inference and Voice Command Construction[24].
Voice Command Feature Extraction takes advantage of DSP algorithms to extract a particular set of coefficients called Mel-frequency Cepstral Coefficients (MFCC) that consist of a signal representation defined starting from a spectrum analysis in a limited amount of time by means of Fourier Transform (DFT). Given the input x[n], it is possible to apply a DFT:

$$X_m = \sum_{n=0}^{N-1} x[n] e^{\frac{-j2\pi nk}{N}} \qquad 0 \leq n < N$$

It is possible thus to define a filter bank M that is used to evaluate the average spectrum around a center of frequency, which varies according to the m$^{\mathrm{th}}$ filter of the bank. The Mel-Frequency Cepstrum is the cosine transform of the logarithm of the magnitude of the M filter outputs, defined as:

$$mfcc_m[n] = \frac{1}{R} \sum_{r=1}^{R} log(MF_m[r]) cos[\frac{2\pi}{R}(r + \frac{1}{2})n]$$

where $\mathrm{MF_m}$ is the mel-spectrum of the m$^{\mathrm{th}}$ frame, defined for r = 1,2,...,R as:

$$MF_m[r] = \frac{1}{A_r} \sum_{k=L_r}^{U_r} |V_r[k] X_m[k]|^2$$

The number of filters is related to both the frequency range under investigation and the sampling frequency of the signal, so fixing m to a certain value is equivalent to limiting the frequency domain of the analyzed frequencies[25]. The reason why MFCC are adopted is related to the implicit structure of the filter bank M that makes them noise resistant. In addition to MFCC, the Voice Command Feature Extraction codelet extracts also their first and second order derivatives, obtaining

thus spectral characteristics that shape the output of this codelet.
After having extracted the spectral features from the input signal, this last has to go through the second step of Voice Command Detection, represented by TensorFlow Inference. The term inference refers to the process of extracting meaningful results from the application of a TensorFlow Lite model on-device in order to make predictions based on input data [26]. TensorFlow is an open source library used for Machine Learning applications characterized by computational speed and lightweight implementation and that can return, for instance, a list of probabilities to be mapped in different categories. This is the goal of the third codelet, the Voice Command Construction one, that makes use of the Command Constructor algorithm to map the list of probabilities coming from the inference output in order to find the most likely sequence of keywords that constitute the robot command [24].
As shown in figure 3.2, this technique allows to identify short commands for user's

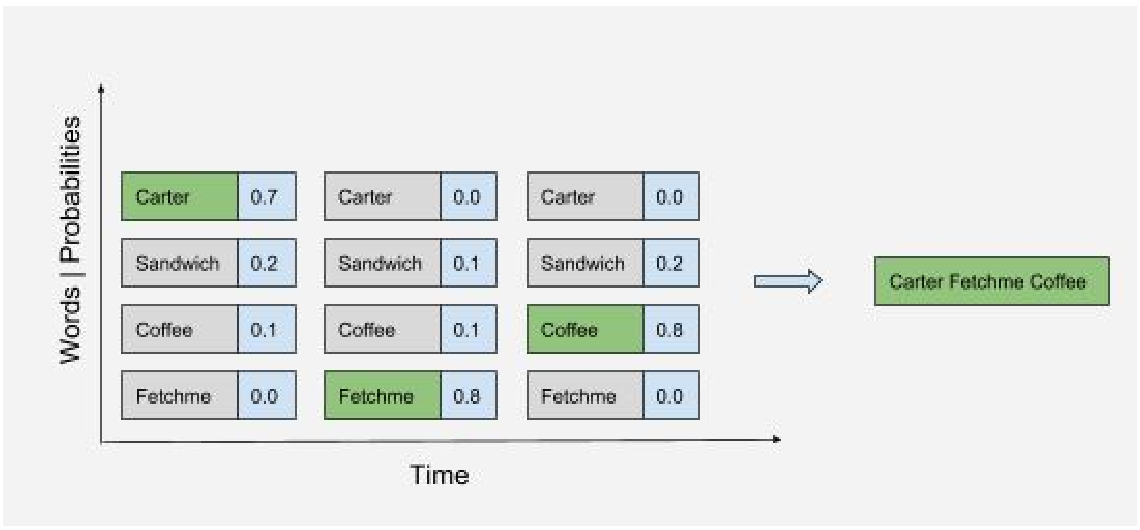


Figure 3.2: Command Constructor algorithm

speech by analyzing the probability that each keyword has been pronounced by the user. Basically the sequence is selected maximizing the sum of all the probabilities for each keyword providing as output the command that has to be converted into an actionable instruction.
After having analyzed the possible Speech Recognition implementation it is possible to say that there are 3 general possible solutions to adopt:

- Integrating existing voice technologies in the app by means of special API or other development tools
- Building an intelligent assistance with open source services
- Creating a custom voice assistance that guarantees a full integration with the app

Moreover the solution to adopt had to be consistent with one of the most stringent design requirement present in the entire development phase, that is to say the absence of a stable internet connection in the airport environment. Adopting a solution that takes advantage of the power of servers for drawing up the audio file coming from the microphone is thus not feasible because it implicitly requires an internet connection. Similarly though, adopting an offline solution like the one used by NVIDIA, requires the purchase of another board as well as the implementation of a suitable communication between the tablet and the Jetson Nano board that involves the usage of time and economical resources. For these reasons, as will be explained in the next sections, it has been chosen to adopt the Google Speech Recognizer API that takes advantage of Google servers, using the Hidden Markov Model algorithm to perform the Automatic Speech Recognition but is also able to work off-line, by previously downloading a word dictionary.

3.2 Voice or Speech Recognition?

Before starting analyzing in detail the basis of HMM algorithm, it is worth to spend some word to briefly explain the subtle but significant difference between two technologies often confused, voice and speech recognition.
Voice recognition is the capability of a device to identify unequivocally a person on the basis of the spoken word input. Indeed voice, on a par with fingerprint represents one of those anatomical parts that are different in each person and so can be used for verification and identification like, as instance, for security purposes in order to give access to private sections. A fundamental feature of voice recognition is that it needs to store the audio file containing the voices to be analyzed because the algorithm compares the input voice with those previously stored inside a database.
Speech recognition instead consists of processing the voice signal, extracting the most relevant features by filtering the noise and converting them into a text on screen or into actionable instructions. It does not require a database of voices but it simply try to translate the command the user is trying to say in a digital record, so it can be used for translations, dictation or automated services like for example virtual assistant or voice control. There are methods to measure the quality of this technology that, as will be explained in the next sections, depends on two main factors, speed and accuracy. The former depends on how efficient is the Hardware-Software architecture that has been adopted for the recognition while the latter depends on how low is the error rate while converting the speech into text [27]. In general terms it is possible to say that although both the two technologies rely on analyzing the same collected data, they provide a different output since they have a complete different final purpose. Indeed, while voice recognition aims only to recognize who is speaking, speech recognition does not care about who is speaking but what is being said, converting the voice into a digital text. For all these reasons just explained it is more correct talking about speech recognition if it is referred to the algorithm designed for the Robotic User Interface.

3.3 Hidden Markov Models

3.3.1 Introduction

One of the problems of greatest significance in the field of engineering is represented by the characterization of a mathematical model for every real-world signal. The knowledge of a mathematical model represents an enormous advantage because it allows to know a priori the behavior of a certain physical signal on the basis of the applied input, learning thus as much as possible about the signal source by means of simulations. In this way the efficiency of the system will be improved without knowing the actual signal.

In general it possible to distinguish between two types of signal models: deterministic models and statistical models[28]. The former is represented by all those signals x[t] in which, given a fixed time instant t_0, the value taken by x[t_0] is a priori known, so in order to determine the behavior of the signal it is necessary to find out its parameters like amplitude, phase, frequency etc. The latter model is represented by all those signals x[t] in which, given a fixed time instant t_0, the value taken by x[t_0] is only describable from a statistical point of view, by means of a stochastic variable and its probability density function. Human voice recognition is one of those real-world signals that can be modeled by means of a statistical model, like the Hidden Markov Model that has been widely used in many applications thanks to its high accuracy.

The first attempts of creating an efficient voice recognition algorithm date back to 1950s with the final aim of creating voice-controllable systems. In 1952 the Bell Laboratories succeeded in developing a system able to recognize numbers from 0 to 9. We have to wait until the '70s before facing with a real voice recognition device created by the Carneige Mellor University that was able to recognize complete sentences even if with a limited dictionary and grammatical structure. It has been immediately figured out the enormous potential of market that a voice-controllable system would have had but the efficiency and reliability of such a device, required very powerful resources and this fact slowed down the arrival of this technology on the market. The development of algorithms that implemented speech recognition had a great boost in the '90s. Nowadays, thanks to the technological progress it is possible to find these types of application everywhere, from smartphones to cars and even in the houses with Google assistant or Amazon Alexa that are able to interact with the user in a natural way, with a very low error rate.

In general speech recognition systems are split in two categories: speaker dependent and speaker independent. The former requires a training session in which the user has to read a short text with normal voice tone and speed, the latter allows the recognition of a generic speaker voice, without any previous training. This implies obviously a lower precision and a different training approach, since the algorithm can not be trained using a single set of voice features but it has to consider different ways to pronounce the same word, creating consequently a bigger database. Although the speaker independent speech recognition requires much more effort from a computational point of view, it is the most used technique in many fields like mobile devices, automatic hot lines or home virtual assistants and it makes use of complex mathematical tools like Hidden Markov Models or hybrid HMM and Artificial Neural Networks (ANN) technologies.

3.3.2 Markov Chains

Before explaining in detail the implementation steps for a Speech Recognizer, it is worth to spend some time talking about Markov Chains that are on the basis for the construction of the Hidden Markov Model. A Markov chain is a model that carries the information about the probabilities of sequences of random variables, namely *states* [29]. In general we can say that a Markov chain is characterized by: a set of N states **Q**, a transition probability matrix **A** in which each component represents the probability of going from state i to state j, and an initial probability distribution π that represents the probability that the Markov chain will start from a certain state [29].
The fundamental assumption on the basis of Markov Chains is the so called **Memoryless property** for which the probability of being in the future state depends only on the previous state, with all the other states that are completely irrelevant. More formally, let us consider a system with N independent states S_1,S_2,...,S_N. As shown in figure 3.3, the states S_1,...,S_5, are described as nodes while the transitions a_{11},...,a_{55}, with their associated probabilities, are described as arcs. For each discrete time instant, a state transition occurs on the basis of the memoryless property, so that the probability of being in the i-th state is given by:

$$a_{ij} = P[q_t = S_j | q_{1..t-1}] = P[q_t = S_j | q_{t-1}]$$

The most relevant characteristic is that the entire process is observable in the sense that each state corresponds to a physical observable event. For those sequences of states in which both observable and hidden events are present, it is necessary to introduce the concept of Hidden Markov Model.

3.3.3 The Hidden Markov Model

A Hidden Markov Model allows to predict the probability of a certain sequence of states considering both observed and hidden events. In the specific case of the implementation of a Speech Recognizer, the observed events are characterized by the words in input while the hidden events are represented by the parts-of-the speech tags [29]. Since HMM are based on discrete Markov chains, they are characterized by the same components to which must be added a sequence of **T** observations taken from a sequence of vocabulary **V**, and a sequence of Observations likelihoods **B** also called emission probabilities. It is important to underline the presence of a second preliminary assumption that characterizes the Hidden Markov Model, namely the **Output Independence**. According to this assumption, the probability of an

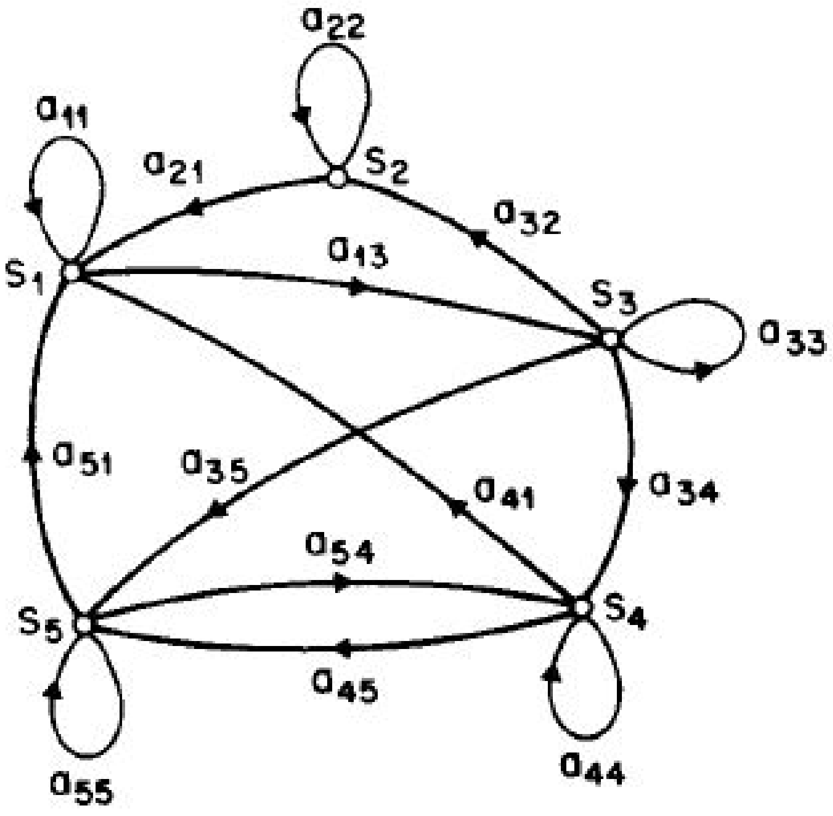


Figure 3.3: Markov Chain example

output observation o_i is conditioned only by the state q_i while all the other states are irrelevant[29]. In formula:

$$P[o_i|q_1...qi, ..., q_T, o_1...oi, ..., o_T] = P[o_i|q_i]$$

As stated by Rabiner in his tutorial [28], there are three problems that have to be solved in order to well define and characterize a Hidden Markov Model:

1. How to compute the probability of the observation sequence O having the model λ characterized by the the transition probability matrix A and the sequence of observation likelihoods B. In a more compact way this problem can be described as the computation of P(O|λ).

2. How to compute the optimal sequence of states Q, having the model λ and the observation sequence O.

3. How to tune the transition probability matrix A and the sequence of observations B in order to maximize the computation of P(O|λ), the probability of the observation sequence O having the model λ.

PROBLEM 1: LIKELIHOOD COMPUTATION

The first problem that has to be solved consists of determining the likelihood P(O|λ), namely the probability that the observation sequence O occurs, considering that model λ has already been established. The solution of this problem in the case of discrete Markov chain is very straightforward, because it only requires computation of the product of all the probabilities associated with the states. The situation is less trivial in the case of a Hidden Markov model because the sequence of the states is unknown, making thus impossible to compute a simple product of probabilities. However, assuming initially to know the observation sequence O and recalling the assumption made before for which the probability of an output observation o_i is conditioned only by the state q_i, it is possible to compute the likelihood of an observation sequence as:

$$P[O|Q] = \prod_{i=1}^{T} P[o_i|q_i]$$

Given the likelihood of a single observation sequence, it is necessary to extend the computation to all the possible sequences, by multiplying each probability for the joint probability of being in specific hidden sequence, generating a specific sequence of observations. Defining the joint probability as:

$$P[O,Q] = P[O|Q] \times P[Q] = \prod_{i=1}^{T} P[o_i|q_i] \times \prod_{i=1}^{T} P[q_i|q_{i-1}]$$

it is then possible to sum over all the possible state sequences, obtaining:

$$P[O] = \sum_{Q} P[O,Q] = \sum_{Q} P[O|Q]P[Q]$$

Although this solution is mathematically correct, it results to be unfeasible from a computational point of view because requires $2T \times N^{\mathrm{T}}$ calculations. This kind of exponential order of magnitude results to be impracticable even with small values of number of states N and number of observations T. Hence it is convenient to adopt a different algorithm called *forward procedure* that has an order of magnitude equal to $N^2 \times T$. In order to facilitate the understanding of this algorithm, the formal expressions will be accompanied by a Trellis diagram example taken from [29]. In a Trellis diagram hidden states are described as circular cells, while observations are described as squares. Each cell α_{t}(j) is the probability of being in state j given the t observations and given the model λ. The value of each cell α_{t}(j) is computed by summing over the probabilities of every path that could lead us to this cell [29]. Formally:

$$\alpha_t[j] = P[o_1, ..., o_t, q_t = j|\lambda] = \sum_{i=1}^{N} \alpha_{t-1}[i]\alpha_{ij}b_j(o_t)$$

where:

- $\alpha_{t-1}[i]$ represents the **forward path probability** computed at the previous time step
- α_{ij} represents the **transition probability** from state q_i to state q_j, i.e. the probability of going from the previous to the current state
- $b_j(o_t)$ represents the **state observation likelihood** i.e. the probability of the observation o_t with respect to the current state j

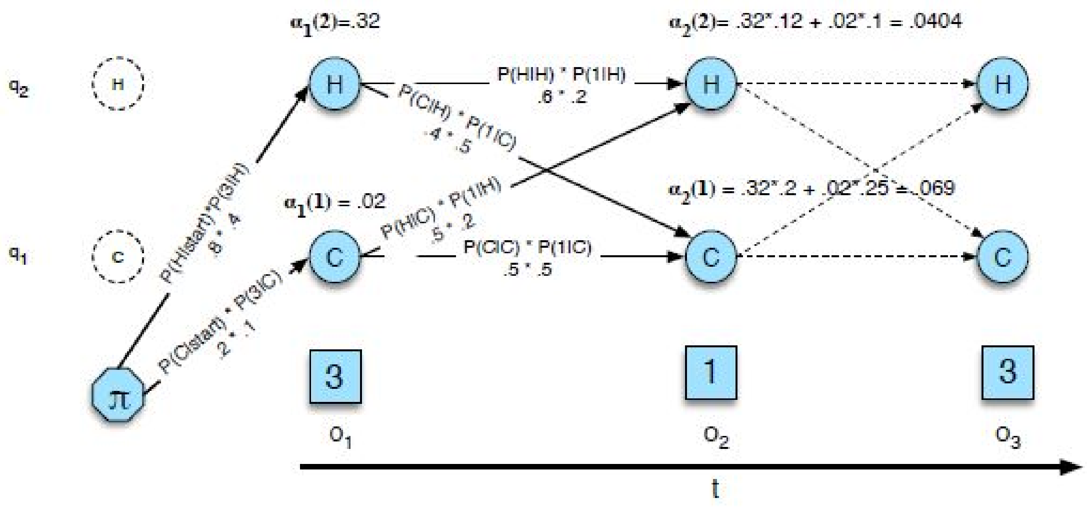


Figure 3.4: Forward algorithm example

The figure 3.4 represents two states, namely H and C and considers a sequence of three observations, o_1=3, o_2=1,o_3=3, By applying the formulas explained above, it is possible to compute easily the value for each cell. Assuming to desire to calculate the probability of being in state q_1 at time instant 2, we obtain that:

$$\alpha_2(1) = \alpha_1(2)P[C|H] \cdot P[1|C] + \alpha_1(1)P[C|C] \cdot P[1|C]$$

where $\alpha_1(2)$ and $\alpha_1(1)$ are the previous forward path probabilities, P[C|H] and P[C|C] represent the transition probabilities from the previous to the current state and P[1|C] represents the state observation likelihood.
The use of this kind of algorithm allows to reduce sharply the computational cost, because if there are N states all the possible state sequences will not be dependent from the observation sequence but will be simply associated to the N states [28].

PROBLEM 2: DECODING PROCEDURE
After having determined the probability that the observation sequence O occurs, considering that model λ has already been established, it is necessary to determine the optimal hidden state sequence knowing the sequence of observations. More formally, the Decoding procedure consists of finding the most probable sequence of

states Q = q_1, q_2,..., q_T, given as input the model λ=(A,B) and a sequence of observations O = o_1, o_2,..., o_T[29].
As in the case of the Likelihood computation, it is not possible to compute the probability of the observation sequence by simply computing the maximum for each possible hidden state sequence because of the computational cost that rises exponentially with the number of possible hidden state sequences. Among all the possible algorithms adopted to find the optimal solution, one of the most famous is the *Viterbi Algorithm* that is a dynamic programming method based on dynamic programming Trellis. In this case each cell v_t(j) of the Trellis diagram does not represent the forward path probability anymore but represents the probability that the model is in state j, given the previous t observations and the model λ computed as the maximum among all the possible state sequences[29]. More formally each value of each cell is computed as:

$$v_t(j) = \max_{q_1,...,q_{t-1}} P[q_1, ..., q_{t-1}, o_1, ..., o_t, q_t = j|\lambda]$$

That can be rewritten in a more compact way as:

$$v_t(j) = \max_{i=1}^{N} v_{t-1}(i) \cdot a_{ij} \cdot b_j(o_t)$$

where:

- $v_{t\text{-}1}$[i] represents the **Previous Viterbi path probability** computed at the previous time step
- a_{ij} represents the **transition probability** from state q_i to state q_j, i.e. the probability of going from the previous to the current state
- $b_j(o_t)$ represents the **state observation likelihood** i.e. the probability of the observation o_t with respect to the current state j

The figure 3.5 shows the same example described during the solution of the first problem but applied to the computation of the decoding task. As it possible to notice, the Viterbi and the forward algorithm are very similar although the computation of Viterbi path probability provides for the replacement of the **sum** for a **max** over the previous computed Viterbi path probabilities. Another fundamental difference between the two procedures is represented by the presence of the so called **Viterbi backtrace**, that is to say a way to keep track of the optimal Viterbi path probability for each time step, in order to easily compute the optimal sequence by going backward on the basis of the stored backpointers.
PROBLEM 3: TRAINING PROCEDURE
The last problem that has to be solved is the tuning of the transition probability matrix A and the sequence of observations B given as input a sequence of observations O and a set of hidden states Q. The most common procedure used for this kind of problem is the **forward-backward** or **Baum-Welch** algorithm that starts

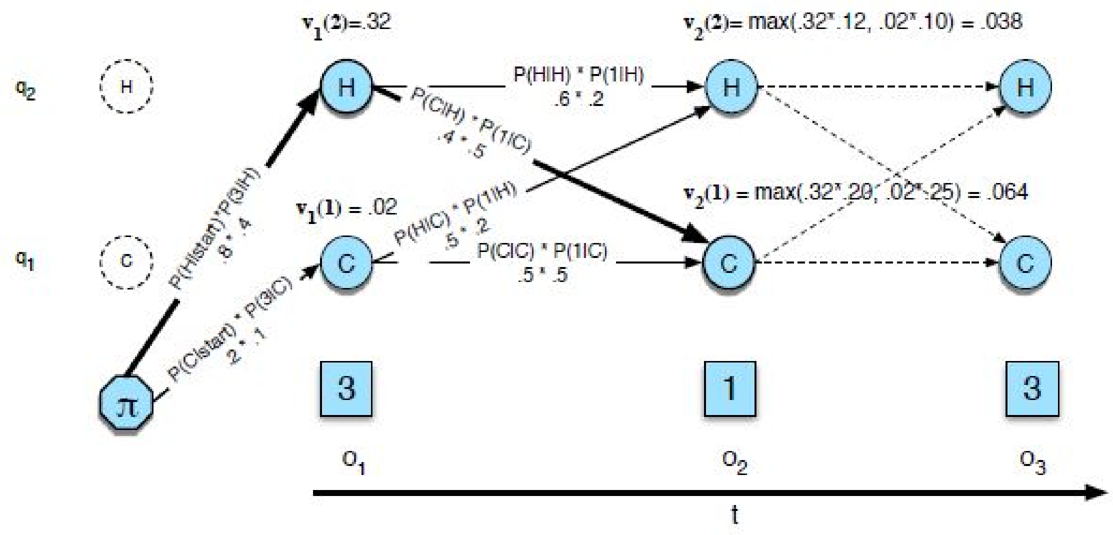


Figure 3.5: Viterbi algorithm example

by estimating the transition and the observation probabilities and then uses these values to improve the quality of the results. Assuming initially to know both the observations and the associated hidden state sequences, it is pretty straightforward to derive the calculations of A and B matrices. Unfortunately, in the real case, it is not possible to determine these counts in a direct way without resorting to iterative algorithms like the one proposed by Baum-Welch in 1972. Basically the algorithm estimates recursively the values of A and B matrices by computing the forward probability for an observation and then dividing that probability mass among all the different paths that contributed to this forward probability[29] as if one applied the Viterbi algorithm twice, first forward and then backward. In order to better understand the algorithm, let us introduce the **backward probability**, defined as:

$$\beta_t(i) = P[o_{t+1}, o_{t+2}, ..., o_T | q_t = i, \lambda]$$

This probability is computed by means of the three steps of initialization, recursion and termination, and together with the forward probability allows to compute the A and B matrices from an observation sequence without knowing the state sequence. Without entering in the details of the mathematical procedures through which the A and B matrices are derived, the final formulas are presented for the two estimations:

$$\hat{a}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \sum_{k=1}^{N} \xi_t(i,k)}$$

$$\hat{b}_j(v_k) = \frac{\sum_{t=1 s.t. O_t = v_k}^{T} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)}$$

where:

- ξ_t(i,j) is defined as the probability of being in state i at time instant t and, at the same time, being in state j at time t+1, provided that the observation sequence and the model are known.
- γ_t(j) is defined as the probability of being in state j at time instant t, provided that the observation sequence and the model are known.
- $\sum_{t=1 s.t. O_t = v_k}^{T}$ is a notation that means *"sum over all t for which the observation at time t was v_k"*[29].

By means of these two expressions, it is possible to perform the re-estimation procedure that is on the basis of the forward-backward algorithm for the tuning of the transition probability matrix A and the sequence of observation likelihoods B. It is possible to state that the forward-backward algorithm is constituted by two main steps, namely **expectation** and **maximization**. During the former, the values of γ and ξ are computed on the basis of the previous estimated values of A and B matrices. During the latter, the values of γ and ξ are used to re-determine the values of A and B matrices. Although this procedure can be carried out without any supervised learning of A and B matrices, in the field of speech recognition it is fundamental the right choice of the initial conditions in order to improve the final estimation.

3.4 Implementation of a Speech Recognition System

The main purpose of a speech recognition system is the conversion of an analog acoustic input coming from the speaker into a set of digital words that can then be associated to actionable commands. This implementation can be done by means of different algorithms like, as instance, Artificial Neural Network, Gaussian Mixture Model or Hidden Markov Model. The latter is still considered the most successful approach to achieve a fast and accurate speech recognition. As shown in fig 3.6,

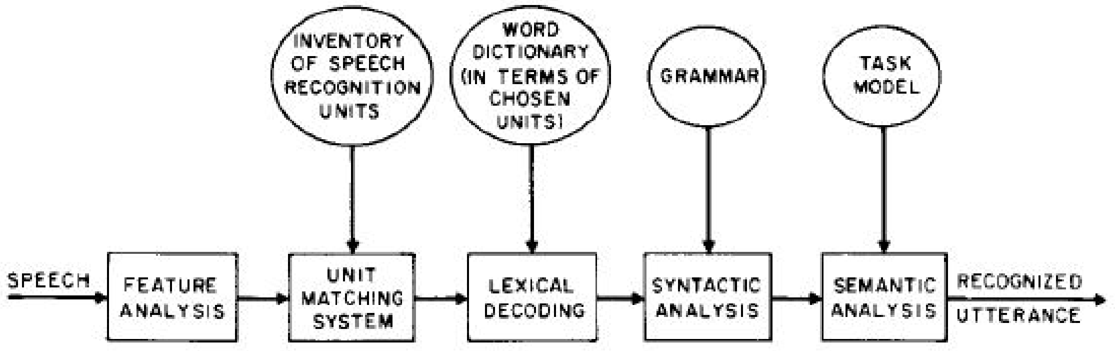


Figure 3.6: Block Diagram of Continuous Speech Recognizer

taken from Rabiner's tutorial [28], the process of a continuous speech recognition system can be partitioned in different blocks:

1. **Feature Analysis**
2. **Unit Matching System**
3. **Lexical Decoding**
4. **Syntactic Analysis**
5. **Semantic Analysis**

The first step of a speech recognition system is the Feature Analysis that consists on extracting equally spaced discrete observation vectors using Mel Frequency Cepstral Coefficient (MFCC). As previously discussed MFCC are a signal representation defined starting from a spectrum analysis in a limited amount of time (typically 20ms) by means of Fourier Transform (DFT). Given the input, it is possible to evaluate the average spectrum using a filter bank and, finally it is possible to extract the observation vectors by using the Cosine transform of the logarithm of the magnitude of the M filter outputs. More specifically, the whole procedure is performed by means of a spectral analysis called **Linear Predictive Coding (LPC)** that consists of seven sub-steps, as shown in figure 3.7 [28]. In the Preemphasis block, the speech signal

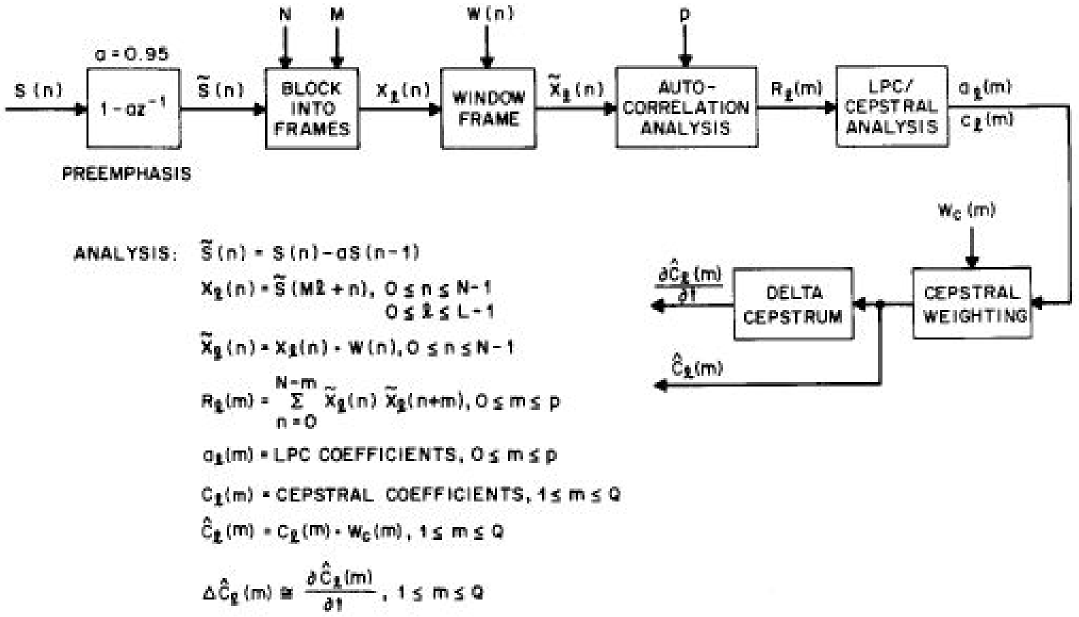

N
M
W(n)
p
a = 0.95
S(n)
1 - az^-1
PREEMPHASIS
S̃(n)
BLOCK INTO FRAMES
X_ℓ(n)
WINDOW FRAME
X̃_ℓ(n)
AUTO-CORRELATION ANALYSIS
R_ℓ(m)
LPC/ CEPSTRAL ANALYSIS
a_ℓ(m)
c_ℓ(m)
W_c(m)
CEPSTRAL WEIGHTING
DELTA CEPSTRUM
∂ĉ_ℓ(m)/∂t
ĉ_ℓ(m)
ANALYSIS:
S̃(n) = S(n) - aS(n-1)
X_ℓ(n) = S̃(Mℓ + n), 0 ≤ n ≤ N-1, 0 ≤ ℓ ≤ L-1
X̃_ℓ(n) = X_ℓ(n) · W(n), 0 ≤ n ≤ N-1
R_ℓ(m) = Σ_{n=0}^{N-m} X̃_ℓ(n) X̃_ℓ(n+m), 0 ≤ m ≤ p
a_ℓ(m) = LPC COEFFICIENTS, 0 ≤ m ≤ p
C_ℓ(m) = CEPSTRAL COEFFICIENTS, 1 ≤ m ≤ Q
Ĉ_ℓ(m) = C_ℓ(m) · W_c(m), 1 ≤ m ≤ Q
ΔĈ_ℓ(m) ≅ ∂Ĉ_ℓ(m)/∂t, 1 ≤ m ≤ Q

Figure 3.7: Block Diagram of LPC procedure

passes through a first-order filter in order to increase the energy of the signal at high frequency and increase the Signal to Noise Ratio (SNR) [30]. The filtered signal is then merged to other N_A speech samples, creating a single frame, while adjacent frames are separated by M_A samples. At this point the frame is windowed in order to minimizing discontinuities, (namely Frame Windowing), and is autocorrelated with a number of coefficients equals to the order of the desired LPC analysis minus 1(Autocorrelation Analysis block) [30]. The last three blocks provide as output the observation vector that is then deployed in the training sessions for creating HMMs associated to the various speech sounds, after having performed a cepstral analysis and a cepstral weighting. Indeed, the observation vector is: *"the concatenation of the weighted cepstral vector and the corresponding weighted delta cepstrum vector"* [28]. The training session is carried out by a Baum-Welch algorithm that iterates the procedure up to the achievement of a suitable convergence interval. Once the discrete observation vectors have been used for the training session, there will be a HMM for each word, with its own parameters A,B and π and the sequence of states represented by the phonemes [31]. The role of the Unit Matching System is thus to provide the likelihoods of a match of all sequences of speech recognition units to the unknown input speech [28]. Lexical Decoding as well as Syntactic Analysis and Semantic Analysis introduce additional constraints to the Unit Matching System block, improving the quality of the matching. More in particular, Lexical Decoding compares the obtained speech recognition units with those present into a word dictionary, in order to understand if the word associated to the model that has been computed, has a concrete sense. If the unit passes the first step, it enters in the

Syntactic analysis block, that compares the general meaning of the word associated with the previous ones, by comparing them with a word grammar. The last obstacle that has to be passed is represented by the Semantic Analysis block that analyses the recognized speech in order to check if it is consistent with the model task. All these three blocks add constraints that improve the general quality of the recognition. Summarizing, for each word that has to be recognized, the first step consists in the measurement of the observation sequence, followed by a detailed analysis of the word structure and by the computation of $P(O|\lambda^v)$, namely the probability of the observation, given a model associated to each word inside the word vocabulary. Once the probability has been computed for each model, the model with the highest likelihood is selected [28]. Formally:

$$v^* = \underset{1 \leq v \leq V}{\text{argmax}}[P(O|\lambda^V)]$$

This computation is usually performed recursively by means of Viterbi algorithm with a computational cost that is consistent with present-day microprocessors.

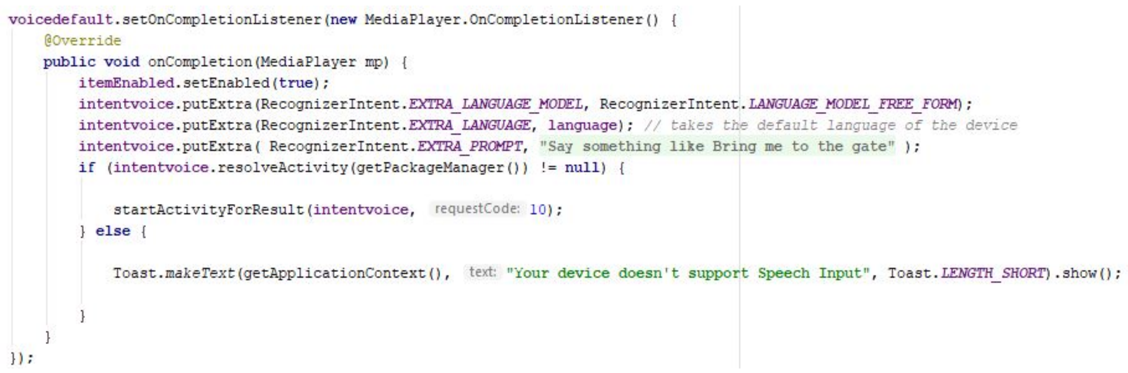
3.4.1 Google Speech Recognizer API

Once the theory about Hidden Markov Model has been deeply examined, a detailed analysis on the practical implementation aspects has to be performed. As already mentioned in the section 2.1, Speech Recognizer API has been chosen because of the stringent requirement concerning the inability of having a stable and reliable internet connection within the airport environment because of the safety rules imposed by the airport security staff. Indeed, this Android based API allows the automatic recognition of spoken language thanks to a word vocabulary available on Google servers but also downloadable for offline uses. This kind of solution brings with it drawbacks like, as instance, the impossibility of using a continuous recognition, which would cause a significant waste of energy and resources as well as bandwidth [32]. This because, as already mentioned during the discussion about Siri, continuous recognition should be deployed on a separated low power Always On processor that can handle the process autonomously leaving the main processor free to carry out other functionalities.
The first step for deploying the Speech Recognizer in the Robotic User Interface consists in adding the **RECORD AUDIO** permission into the Android Manifest in order to allow the usage of the microphone resource:

<uses-permission android:name="android.permission.RECORD_AUDIO">

After that we need to create an intent instance for the recognizer, since the Android speech recognizer has been created in the form of an intent, i.e. a request from a component of a feature that is implemented in another component. It is used to make the applications interact with each other, both those programmed and

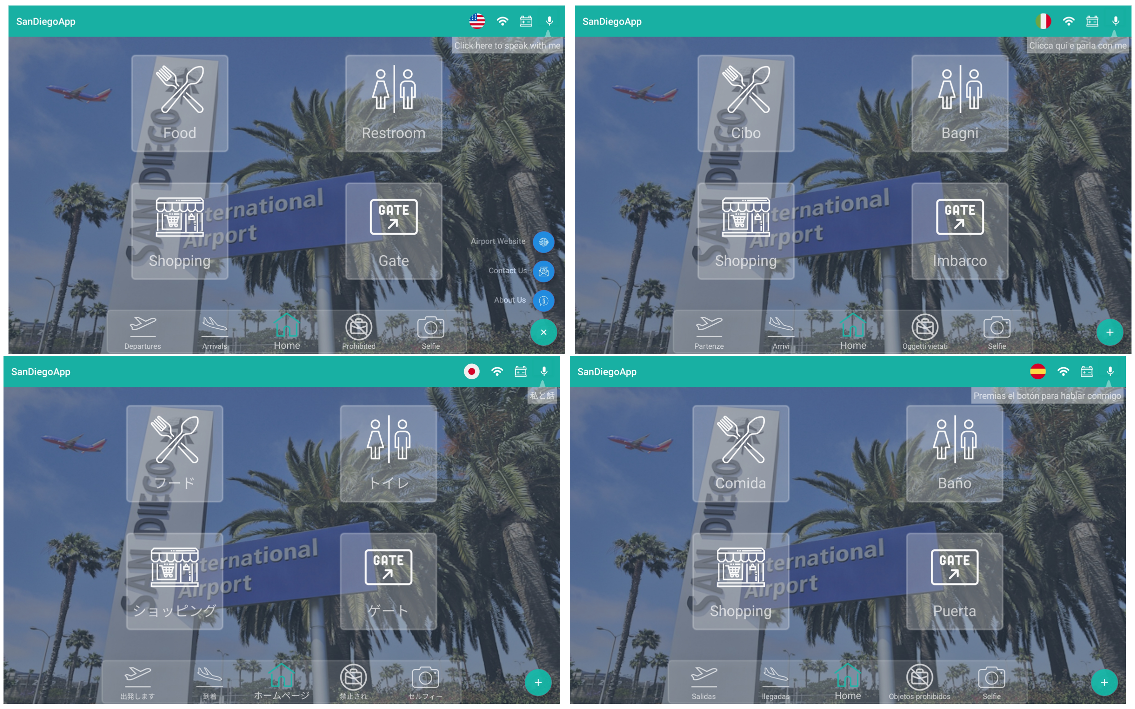
those already present in the system. The instance is called by means of the following command:
Intent intentvoice = new Intent(RecognizerIntent.ACTION_RECOGNIZE_SPEECH);
Every time the user taps on the microphone icon the Speech Recognizer is triggered. From a coding point of view, this means that the Speech Recognizer intent has to be enabled inside the *onOptionsItemSelected()* method and has to listen to the speaker to capture the input to be analyzed. It is worth to notice that in this case the method triggered when the microphone icon is clicked is not the usual *setOnClickListener()* because the microphone is part of the set of toolbar items. As it is possible to see



```
voicedefault.setOnCompletionListener(new MediaPlayer.OnCompletionListener() {
    @Override
    public void onCompletion(MediaPlayer mp) {
        itemEnabled.setEnabled(true);
        intentvoice.putExtra(RecognizerIntent.EXTRA_LANGUAGE_MODEL, RecognizerIntent.LANGUAGE_MODEL_FREE_FORM);
        intentvoice.putExtra(RecognizerIntent.EXTRA_LANGUAGE, language); // takes the default language of the device
        intentvoice.putExtra( RecognizerIntent.EXTRA_PROMPT, "Say something like Bring me to the gate" );
        if (intentvoice.resolveActivity(getPackageManager()) != null) {

            startActivityForResult(intentvoice,  requestCode: 10);
        } else {

            Toast.makeText(getApplicationContext(),  text: "Your device doesn't support Speech Input", Toast.LENGTH_SHORT).show();

        }
    }
});
```

Figure 3.8: Speech Recognizer piece of Software

in figure 3.8, inside the *onOptionsItemSelected()* method, another method is called, namely *setOnCompletionListener()*, that starts a default voice able to interact with the user. This voice is programmed to ask the user how it can help and, once the audio is finished, it calls again the Speech Recognizer intent for receiving the second input audio to be examined. During this phase the **language** parameter is taken by the intent in order to reproduce the audio file in the correct language. Indeed, as already mentioned in the previous chapters, five languages have been implemented: English, French, Italian, Spanish and Japanese. The user can click on the flag icon to change the labels of the whole interface as well as the language of the virtual assistant, so this information has to be stored in order to program the virtual assistant accordingly.



Once the language has been selected (figure 2.12), the interface presents the homepage accordingly and the virtual assistant starts speaking in the language consistent with the one selected by the user. Once the user has pronounced the command, this is extracted and analyzed by the speech recognizer according to the rules explained in the previous paragraph. Each word is thus selected on the basis of its matching likelihood and the final sequence is provided as output to the method *onActivityResult()*. This method compares the output sequence with some pre-selected commands and in case of match, it triggers a consistent actionable instruction. If, as instance, the sequence provided by the Speech Recognizer contains the word *"Gate"*, the Gate activity is triggered and the virtual assistant asks the user which kind of gate he wants to reach. By simply downloading the word dictionary for every supported language, it has been possible to implement an offline Speech Recognition system, with the minimum amount of resources obtaining at the same time reasonable accuracy with respect to the speed of the conversion process.

Chapter 4

Robot Communication Systems

Summary

This chapter discusses more in detail the existing communication systems among robot subsystems trying to carry out a specific analysis about all the protocols adopted for a communication system between the Robotic User Interface and the other subsystems. After having analyzed in detail all the possible solutions, the proposed one will be explained putting on evidence pros and cons from both a technical and an economical point of view. Afterwards, the interaction with the Airport Server will be examined, explaining all the key concepts useful to understand the remote connection and its advantages in terms of cybersecurity. Finally an analysis among all the parsers will be conducted to better understand how to read data from a web server in the easiest way.

4.1 State of the art and design requirements

Communication plays a fundamental role in service robot systems and can be defined as a protocol through which it is possible to easily interface the robot with the external world. A service robot can establish the communication in several different ways, like:

- Between a robot and a robot user interface, to send commands from the user to the robot and to receive back status information, data relevant to the robot mission and environmental data from the robot system.
- Between more than one robots in a multi-robot system to support local sensor data fusion, to support both planning and coordinated execution of cooperative behaviors between robots, and/or to allow one robot to serve as a communications relay for other robots and/or user stations
- Between the robot and external clients, transferring data coming from sensors or processed information

- Between robots and the robotic system developer, to reduce technical risk and increase implementation productivity by providing software downloading and system debugging tools to exercise and validate hardware as well as software [33].

Considering more in general the robotic communication system, as explained by professors Dennis Krupke, Jianwei Zhang and Frank Steinicke in their article about Mixed Reality Robotic User Interface, the general structure of information flow in a robot communication system is constituted by human, robot and world as the essential entities of a human-robot interaction task [34]. This basic scheme, as shown in

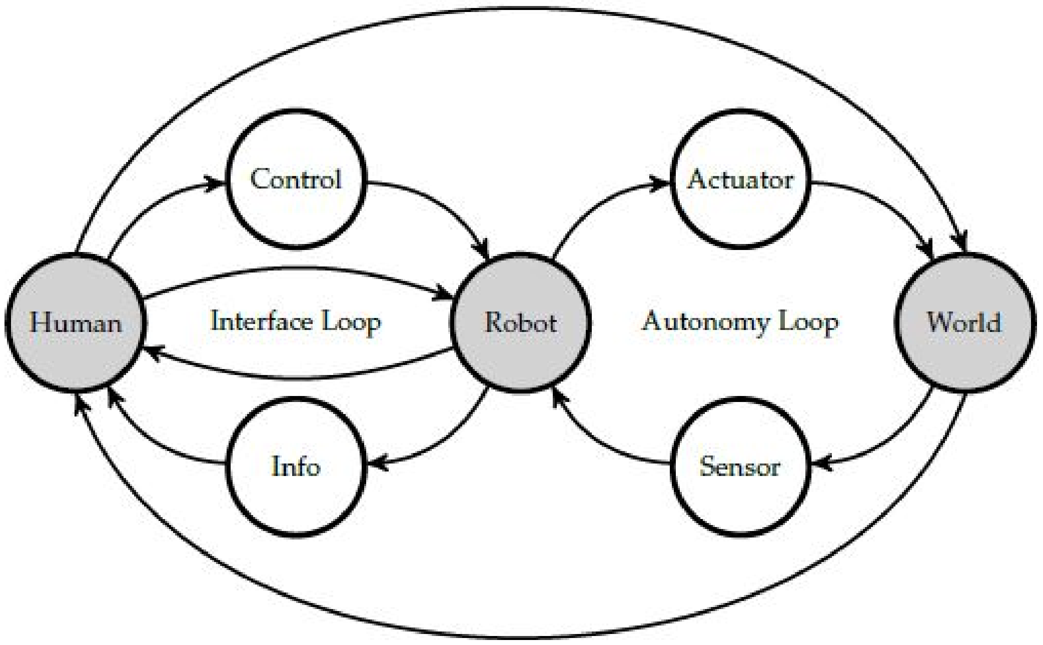


Figure 4.1: Robot Communication Scheme

figure 4.1, explains the way through which the general communication is carried out: the human entity transfers control data to the robot and receives some information as in a feedback loop. Once the robot receives the command by the human entity, it interacts physically with the real world by means of actuators and sensors and it can do this by means of an autonomy loop as in the case of the designed robot thanks to visual SLAM algorithm and its odometry. The knowledge of the robot about the world and its own state will be accessible for the human operator by the robotic user interface, represented by the tablet. As it is possible to notice on the above scheme, the whole communication system is not only constituted by the robotic user interface, already analyzed in chapter 2, but also by a series of hidden layers that allow the robot to communicate with the human operator and, at the same time,

to be informed about what is going on in the real world. In the particular case of the designed robot, these hidden layers are represented by sensors like Stereo camera, Lidars, encoders, but also by Jetson TX2 board, F28069-LAUNCHXL board, CC32220 Wi-fi module and Robot Control Unit. All together they compose the back-end of the robot, the side that does not interface directly with the human entity but that has a strong impact with the entire communication process. After this brief elucidation about the communication system a case study has been therefore conducted about most recent stages in the development of communication protocols for a Robotic User Interface, although all the companies already mentioned like Savioke, LG or Aethon have not provided all the technical information for a matter of corporate privacy.
Among all the analyzed communication system protocols adopted on service robots, it has been possible to distinguish between two macro categories: wired and wireless communication systems. In general the world is increasingly going wireless and this is even more true for autonomous service robots where the presence of wires could compromise the mobility itself. However, wired communications have some advantages that can not be ignored, like speed and reliability of data transfer and in particular the security aspect that, in an area such as the airport one, takes a role more important than ever. In fact thanks to the fact that devices can access the wired network over an improved control, the probability to tackle with malfunctions caused by malware connecting to the network is really low. For these reasons, the majority of the scientific paper suggests the adoption of wired communication among the subsystems of the same robot and a wireless solution for the interaction with the external world. An example of examined application is represented by a decentralized traffic control system that increases driving throughput and driving safety by providing automated cooperation between vehicles, decentralized train traffic control, and cooperation of mobile robots [35]. As it is possible to see in figure 4.2, the

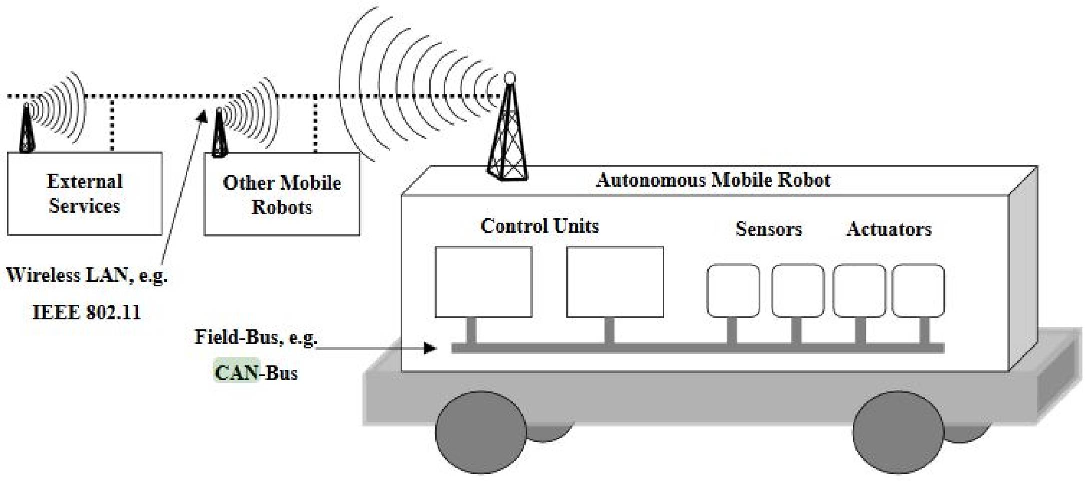


Figure 4.2: Robot Communication Network

Table 4.1: CAN messages structure

TYPOLOGY	**DESCRIPTION**
Data Frame	It contains data to transfer from one node to one or more receiver.
Remote Frame	It is used by a node to request the transmission with the same identifier. It implements the RTR (Remote Transmit Request) mode
Error Frame	It is transmitted when an error occurs.
Overload Frame	It is used to increase the delay between a Remote Frame and the following one.

autonomous robot is characterized by a transmitter and a receiver that, thanks to a wireless LAN protocol allow the robot to communicate with external services or other mobile robots. Internally we have Robot Control Units, sensors and actuators, all connected by a Field-bus like a CAN-bus, one of the most used standard for wired communications. CAN (Controlled Area Network) bus is a *twisted pair multidrop* wire with a maximum payload of 8 byte. The messages sent by means of this protocol are characterized by the so called *Cyclic Redundancy Check*, to reduce the error occurrence, and by an *identifier* that is used by every node to decide if the message just read has to be turned into a specific action or has to be ignored. Every bit transmitted is defined as *recessive* or *dominant* and thanks to this implementation, every node can transmit and receive at the same time without any risk. In fact, in case of simultaneous transmissions from two or more nodes, the bits contained in the message will be compared one by one ignoring all the message whose bits are recessive. At the end of the comparison only one among all the compared messages will be send. So thanks to this smart solution, called *nondestructive bus arbitration*, in case of simultaneous transmissions, the message with the highest priority will not be destroyed but it will be always transmitted, improving the quality and the speed of the transmission. In the table 4.1 the general structure of a CAN message is shown.
Every Data Frame or Remote Frame is separated from the next one by a delay called *Interframe Space* that has the role of letting the other nodes to understand that the transmitted message has been completed. These just explained reasons, together with a very low cost of the associated hardware, have led the majority of the companies to choose the CAN bus as the standard protocol for the communication between the robot subsystems. Despite of the importance of this protocol inside the robot communication system, it will not be analysed more in detail since the Robotic User Interface is characterized by an hardware that does not support the CAN protocol.
As regards the external world interaction of the analyzed robot, the wireless LAN

protocol has been chosen, selecting the standard IEEE 802.11, which defines the physical layer and the medium access control in the data link layer for wireless local area networks [35]. Before starting focusing in detail all the possible robotic interactions a premise has to be made about the fact that it has been chosen to run the Interface on a separate hardware device with a separate processor with respect to the robot main brain, that has its own software ecosystem. In fact while the entire SLAM and object detection algorithm is run on the Jetson TX2, and is coded in C++, the selected tablet supports Android language, that is to say Java language. For these reasons, in the following pages will be presented only the communication protocols that guarantee their compatibility with both C++ and Java.

4.2 Interaction protocols among Robot Subsystems

Before examining all the different protocols, it is worth to say a few words about the layers that reflect the various aspects of communication. The standard ISO introduced seven levels of understanding, each with its related rule but the most used ones are only four levels: physical, data link, network, application.
The physical layer consists on the physical specifications of the connection like physical wire or type of transmitted signals. It is possible to establish a communication by means of two different techniques that are the actual physical wire like RS232 port and the radio communication. The former is one of the most famous serial communication interface, introduced in 1960 and still widely used. Even in the robot prototype designed for running tests the RS232 port has been used for connecting the narrow FOV Lidars with the F28069-LAUNCHXL board, as shown in figure 1.7 in section 1.2. RS232 interface specifies electrical levels of the signals to be used, the type of connector (figure 4.3), the wires for controlling a modem, useful for the connection in case of distances greater than 15m and three wires for transmitting and receiving digital signals, called transmit data (TXD), receive data (RXD) and ground (GND). It is worth to say that, in case of distances less than 15m, it possible to ignore the modem, and connect just the wires 2,3 and 5 as shown in table 4.2 by avoiding the use of the handshake mode.

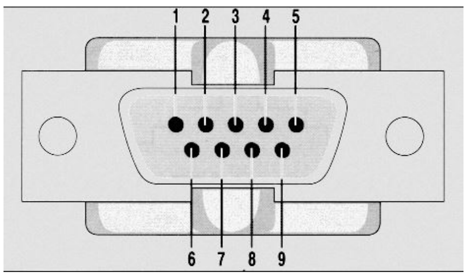


Figure 4.3: RS232 Connector

PIN	SIGNAL
1	Data Carrier Detect
2	Received Data
3	Transmitted Data
4	Data Terminal Ready
5	Signal Ground
6	Data Set Ready
7	Request to Send
8	Clear to Send
9	Ring Indicator

Table 4.2: RS232 PinOut

The second level, the data link one, is in charge of managing the communication between receiver and transmitter, by setting suitable transmission rules. This level does not deal with the interpretation of the content of the message but only with correct shipment of the message itself. In order to do this, a preliminary structure has to be chosen for the robot communication like, as instance, a Master/Slave structure in which parameters like identifier, Cyclic Redundancy Check code, message or effective number of word characters have to be shared between receiver and transmitter.

The Network level, the third one, provides methods for creating a virtual communication for transmitting data from node to node. There exist several methods of this layer, like internet working, error handling or packet sequencing.
The last level, the Application one, is the one through which it is attempted to understand the content of the message. The management of this layer is the most complex one because, as it can be easily understood, the message changes from time to time. Since the Application level is on top of all the others, it does not provide services to the other levels but at the contrary, it directly interacts with the applications used by the user, providing network services. It is possible to say that it basically provides an interface useful to give the user access to the network, like in the case of the HTTP protocol.
Once the preliminary analysis on the general rules about the communication protocol has been made, it is possible to going deep in all the protocols that are compatible with both C++ and Android and so, as a matter of principle, applicable to the robot to be designed.

4.2.1 USB Protocol

USB, acronym that stands for Universal Serial Bus, is the most common standard for the connectivity of PC peripherals. On the basis of the USB protocol there is the concept, usually unconsidered, that during a connection there is always a device considered *host* that acts as a master, and another device considered *peripheral* that acts as a slave. Thanks to this kind of implementation it is possible to establish a communication without running a risk of possible conflicts. When a device is connected to the USB host for the first time, a configuration process is launched through which the USB host requests to be identifiable by means of its vendor ID and its product ID. This process is usually known as drivers installation. Once all the drivers have been installed, the communication can be carried out, with a data transfer speed that depends on the wire that has been used. The table 4.3 provides a series of terms useful for better understanding the structure of the communication:
The communication is made by means of a differential implementation, that is to say that a wire carries the signal and the other one carries the inverse of the same signal. In this way the sum of the voltage on the two wires will be always constant and the receiver will read the difference between the two signals, obtaining data transfers purer, immune from errors and noise.
From the physical layer point of view the USB connector is based on four pins, two for data transfer, one for the ground and one for the power supply, as it is possible to see in figure 4.4. This kind of physical configuration has been the reason why USB became the common standard for peripherals communication. In fact it guarantees a total cost really modest thanks to the low number of pins, but also to the differential communication that offers a fast and reliable data transfer and, moreover, thanks to the presence of ground and supply wires that allows to use the USB cable not only as data transfer but also as power supply for the peripherals.

Table 4.3: USB Key concepts

CONCEPT	**DESCRIPTION**
Host	The device in the USB link that commands or ”masters” the communication
Peripheral	The device in the USB link that listens or acts as a ”slave” in the communication
Enumeration	Connection process between USB Host and peripheral devices
Driver	Software installed on Host PC to tell the Host how to communicate
Device	Any physical hardware that has data endpoints for USB data transfer
Endpoint	A source or termination of USB data.

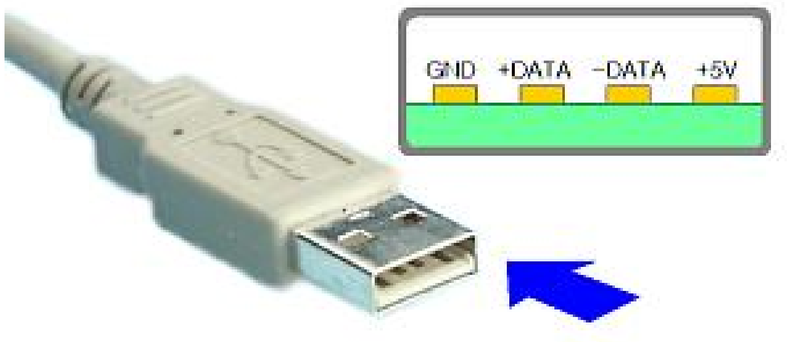


Figure 4.4: USB connector Pin Out

From the data level point of view it is possible to notice that the USB protocol makes use of a particular encryption called NRZI *No Return Zero Inverted*. Assume to call A and B the values of the lines status. In the USB communication the bit is coded as ’1’ if the A or B status of the line does not change between two successive bits otherwise it will be coded as ’0’. In this way, the status of the line will change for every bit when a sequence of ’0’ wants to be transmitted, allowing the receiver to be easily synchronized. In order to guarantee synchronism, no more than six bits ’1’ have to be transmitted because the receiver needs a bit ’0’ to have a reference on the line. In this mechanism, the receiver is able to understand autonomously that a bit 0 has been sent just for synchronism, so it can ignore him in the communication process.
For all these reasons the USB protocol has been analysed in order to understand advantages and disadvantages of its use in the communication between the tablet

and the Jetson TX2 board. An important feature that has been introduced starting from the USB 2.0 is the USB On The Go (OTG) that let the peripheral (the tablet) to perform some functions usually performed only by the master, letting a point-to-point connection even in absence of an host. In order to try the effectiveness of this type of serial communication an Android application has been developed introducing in the Android Manifest the command:

<uses-feature android:name="android.hardware.usb.host" >

to enable the USB Host mode. Once inserted the unique vendor ID of the Jetson TX2 board to establish the communication, the *onResume()* method has been called which called another method that scans all the USB devices looking for the one with the suitable vendor ID. At this point the first disadvantage of the use of this protocol has been discovered, that is to say that the scansion process to connect the host device is not made automatically but requires the intervention of the user. Once the device has been found, it is classified as a *Communications Device Class* i.e. a device characterized by multiple communication interfaces. In this kind of context the suitable interface is based on two end points with the exit one that supports a *bulk* data transfer i.e. a data transfer in which data are compressed, blocked and buffered in order to optimize the process. Once the interface has been identified, a communication channel is opened towards the device and host, so that the client can start sending and receiving messages. From a data link level point of view, the structure of the message is based on a first parameter for the request with an associated value of 0x20, and an array of seven bytes in which 4 bytes are used for the baud rate, 1 byte for the stop bit, 1 byte for the parity bit and 1 byte for the word dimension. Performing some tests it has been noticed that the data transfer was subject to periodical errors after 3-4 transmissions because of a problem known as *Buffer Overflow Exception*. Basically the Bulk mode transfer makes use of the same buffer resource even if it is saturated creating the problem of the overflow. This reason, coupled with the non-appealing presence of a physical wiring and with the problem of configuring manually the connection between the two devices has brought the company to leave aside this kind of solution with respect to a wireless one.

4.2.2 Bluetooth Protocol

The term Bluetooth indicates a network technology that has been raised to industrial standard for radio communication by the IEEE 802.15.1 group of the US Institute of Electrical and Electronics Engineers. Bluetooth is used for wireless point-to-point transmission and voice and data connection between two separate digital devices. The main objective of the technology is to replace, or make completely obsolete, the connections with wires particularly for mobile devices such as smartphones and tablets where wires represent a physical limitation. An interesting fact concerns the choice of the name. In fact it has been chosen in honor of a Swedish king Harald Gormsson called *Bltand* i.e. blue-tooth that succeeded in unifying several clans under a central government as well as this protocol was born with the final mission of unifying all the transmission protocols into a unique standard. Besides the story of the name, the most important characteristic of this protocol is that, compared to other technologies such as USB, LAN or WLAN, it is used for data transfer over a short distance and is characterized by the simplicity and efficiency of the way through which it establishes the connection. With respect to the other technologies mentioned before, it reaches lower transmission speeds, so that sending larger amounts of data may take longer but for the transmission of single files and less complex uses like the simple messages that had to be exchanged between Jetson TX2 board and Robotic User Interface, Bluetooth can be considered undoubtedly a valid solution. For these reasons this technology has been analysed in detail as a possible solution for the interaction among robot subsystems. From the implementation point of view, in order for a digital device to become Bluetooth-compatible, it must have a specific data transfer control software and a particular Bluetooth chip equipped with a transmitting unit and a receiver installed in the hardware components. The Bluetooth frequency falls within the non-licensed ISM band in the range from 2.402 GHz to 2.480 GHz. Compatible devices that comply with the Bluetooth SIG standards can, as Short Range Devices (SRD), operate without restrictions worldwide. Each device, to identify itself, has an individual 48-bit MAC address. As in the case of the USB protocol, a Bluetooth connection can be started from any device that assumes the role of *Master* with respect to a *Slave*, creating a so-called *"piconet"* (a Bluetooth network). The connection remains active until the master deactivates the Bluetooth function in its operating system. Devices that want to connect to a piconet, listen to the Master signal in the scan mode, with intervals of 2.56 seconds even if the connection is generally established within 1.28 seconds. The connection of two or more devices via Bluetooth is also called *"pairing"* [36]. From a practical point of view it is necessary that the distance between the participants of a piconet is reduced and that the Bluetooth function of the respective devices is active. The activation takes place, depending on the device, through a specific software, a check box or a button, marked with the Bluetooth symbol. The connection must then be authorized by means of a PIN (often with four digits) which is displayed on the screen of the Slave device or is shown in the respective manual. This access procedure is intended to guarantee security to third parties and should generally only be performed once. The paired device is saved in a list and automatically connects as soon as it enters the range of the piconet, provided that Bluetooth is activated. After having analyzed all the theoretical aspects of this protocol, an Android project has been developed to test the feasibility of the communication. The first thing that has been done was to add the Bluetooth permissions inside the Android Manifest:

```
<uses-permission android:name="android.permission.BLUETOOTH">
```

<uses-permission android:name="android.permission.BLUETOOTH_ADMIN">

After that, two methods have been implemented: *receiveMessageBluetooth()* that is called every time the host (represented by the Jetson TX2 board) wants to send a message to the client (represented by the tablet); *sendMessageBluetooth()* that is called every time a button has been clicked on the screen. This latter method has been implemented to send manually a message by tapping a button on the screen. Basically the click on the button triggers the method *setOnClickListener()* that in its turn triggers the method *sendMessageBluetooth()*. After having tried this simple application, it has been noticed that even if the messages sent were very light, the communication turned out to be slow and some time even interrupted. For these reasons other protocols have been studied in order to set a suitable real-time communication without lags and random interruptions very difficult to predict.

4.2.3 MQTT Protocol and Eclipse Paho

The last protocol that has been analyzed was the MQTT one. As stated by the official MQTT 3.1.1 specification, MQTT is "a publish/subscribe extremely simple lightweight messaging transport protocol, designed for constrained devices and low-bandwidth, high-latency or unreliable networks" [37]. The term MQTT stands for MQ Telemetry Transport, where MQ represents a series of products developed by IBM to support telemetry transport. Indeed, this protocol has been brought on the market in 1990 by IBM, with the main mission of monitoring a long oil pipeline in the desert. In such an environment practically devoid of infrastructures, it was necessary to establish a light connection without spending too much money in energy consumption but that guaranteed at the same time a stable communication. These were the historical reasons that led IBM to invest in such a technology that nowadays is widely used especially in Internet Of Things applications and that, with its last version, it is even become a standard. In contrast to HTTP that, as will be explained, is based on a client-server communication principle, MQTT is characterized by a different mechanism in which a publisher publishes one or more messages on a specific topic and one or more subscribers that can receive the message. A fundamental role is then played by the broker that manages the communication sorting the messages on the basis of their topic. In other words, this protocol is not peer-to-peer because a message can be published even without an explicit request by the client. MQTT protocol is based on the architecture shown in figure 4.5 taken from HiveMQ official website [38]. As it is possible to see in the example, a Publisher represented by a speed meter publishes the speed measurement on the topic **speed**. This message is not sent directly to the subscribers but is sent to the broker that has the mission of receiving data on one or more topics, sorting and sending them only to the subscribers that have submitted to that specific topic. Once a subscriber has been undersigned to a particular topic, as soon as a message on that topic is available on the broker level, it is sent almost immediately with a

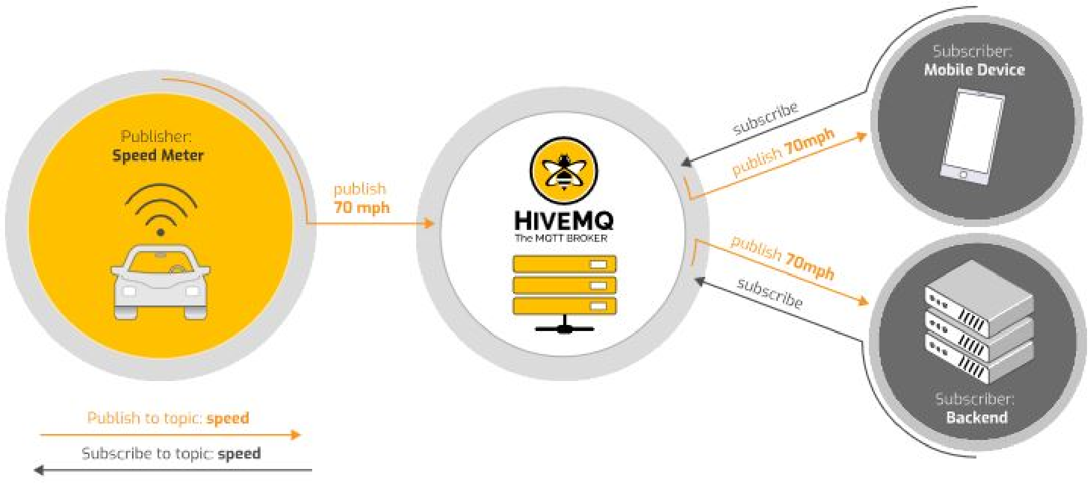


Figure 4.5: MQTT Architecture

very low latency. In order to summarize it, is possible to say that MQTT implements natively a TCP socket based on the three following elements:

- BROKER that is the core of the communication and it is in charge of dispatching all messages between the senders and the rightful receivers
- PUBLISHER that is in charge of sending messages associated to a topic no matter if one or more subscribers are actually listening to them
- SUBSCRIBER that is in charge of receiving the messages associated to the topic to which it is undersigned.

It is possible to notice that this kind of architecture is decoupled from three different points of view. There is a spacial decoupling due to the fact that both publishers and subscribers do not need to know the actual presence of the other one in order to establish a communication, since they only make reference to the broker. There is also a temporal decoupling since it is possible to have a feasible communication even if publisher and subscriber are not connected at the same time, provided that they are referred to the same broker and to the same topic. Finally there is a synchronized decoupling since the subscriber receives the messages associated to the topic which it is signed up for, as soon as it is connected to the broker.
A central role in the MQTT paradigm is played by the **topics**, routing information written as UTF-8 strings based on more hierarchy levels. The levels are separated by means of a slash. Assuming, as instance, that a subscriber decides to undersign to the topic *home/livingroom/temperature*, it means that it will receive temperature measurements associated with the specific temperature sensor present in the living room of the specific house. There are other special symbols like + that is a wild card

Table 4.4: QOS levels

QOS	EXPLAINATION
0	There is no quality. The publisher is limited to send messages without having guarantees of being received.
1	The quality is improved by means of an acknowledgement that has to be transmitted within a certain time. It means that the subscriber could receive multiple messages if does not send the acknowledgement back because the subscriber thinks the message has not been transmitted correctly.
2	The highest level of quality. Single delivery is guaranteed by means of the implementation of a 4 steps handshake protocol.

single level. It is used when the subscriber wants to receive information about an entire level of the hierarchy. For example the topic *home/+/temperature* means that the subscriber wants to receive the information about temperature measurements coming from all the sensors in that specific house. In contrast to the + symbol there is the # symbol that is a wild card multilevel. When a topic presents that symbol like for instance *home/#*, it means that every measurement coming from every sensor in the house has to be read.
As already explained the MQTT protocol is spatially and temporally decoupled, so it is not possible to understand if the messages that have been transmitted by the publisher will never be read by some subscriber. In order to solve this problem, the Quality Of Service (QOS) concept has been introduced in the paradigm. QOS has been defined with reference to a single message, so it can happen that the same publisher publishes messages with different QOS levels depending on the need. The possible QOS levels are managed by the broker and they represent an additional service above the one offered by the TCP socket.
As shown in table 4.4, there are three QOS levels, that guarantee an higher and higher quality of the communication, having the assurance that the messages will be delivered. Besides the implementation of the QOS levels, there are other mechanisms for the management of this pattern publisher-subscriber. One of the most used is the Last Will and Testament message (LWT). This is an example of message set by the publisher when it is connected to the broker on a particular topic. This message is not re-published by the broker unless the client does not disconnect in a sudden and unexpected way. In that case, the LWT message will be send out to all the subscribers on the line that will decide consequently how to react, for example disconnecting and undersigning to another topic. Another common mechanism is represented by the Retained messages. When a message is marked as retained, its last value is stored in broker memory so that, whatever subscriber decides to submit to that particular topic even if with a certain delay, it will be able to receive the last

value of the topic to which it was submitted. In other words it can be said that a sort of virtual buffering is implemented every time a message is marked as retained. The last analysed mechanism is represented by the clean and the durable sessions. If clean session is set to false, then the connection is treated as durable: when the client disconnects, any subscriptions will remain and any subsequent QOS 1 or 2 messages will be stored until it connects again in the future. If the clean session is instead set to true, all the subscriptions will be removed for the client when it disconnects. It worth to say that durable sessions are a powerful tool that works if and only if the publisher re-connects with the very same credentials that it had when it was connected the first time, otherwise the broker will treat it as a new publisher and it can not have access to the old session.
Once the theory about this protocol has been deeply examined, a detailed analysis on the implementation aspects has to be done. As regards the broker, it is very uncommon to design it from scratch but it is preferable to use already existing software. Some of the most commonly used MQTT broker are Eclipse Mosquitto, HiveMQ and IBM WebSphereMQ. In particular Eclipse Mosquitto is an open source broker written in C and with the peculiar aspect of being very lightweight so particularly suitable for constrained devices. Because of all the already discussed problems with San Diego airport security, the only solution that has been considered feasible was to establish a MQTT communication without an internet connection but performing everything by exploiting a local host 192.168.43.1 on the only available port, the 1883 one. For this reason, an Android mobile device has been added during the tests, with a Wi-Fi hotspot turned on so that both tablet and Jetson TX2 board could have established a connection. After having set all the parameters related to the broker, the MQTT client library has been chosen. Among all the libraries, Eclipse Paho has been selected since it represents the most popular client library implementation, available for several programming languages. The step by step implementation of the communication between Robotic User Interface represented by the tablet and the Robot subsystem represented by the Jetson TX2 board is the following:

1. The customer selects one destination among the options
2. The app sends the selected destination to the Jetson
3. The Jetson confirms the destination and sends a confirmation message back to the app
4. The app shows a confirmation message and requests the customer if he is ready or not
5. Once the customer is ready, the app sends this information to the Jetson
6. The SLAM software sends the ”forward/reverse” command to F28069 (red board)

7. When the robot arrives, the SLAM sends the "arrived" message to the app
8. The app plays the "arrived" message informing the customer that the destination has been reached
9. An approval rating survey pops up asking the customer to rate the experience
10. The SLAM drives back the robot to its initial point.

As it is possible to notice, both Jetson and tablet have to act as Client but also as Subscriber depending on whether they are sending or receiving a message, so it has been necessary to develop both the cases. In the next pages, just the publisher/subscriber implementation on the tablet will be examined. In the activity called *MQTT Activity*, the communication protocol for as regard the Robotic User Interface part has been designed. First thing that has been done was to sort out dependencies which are libraries needed to setup an MQTT client and service in an Android app. As previously announced the Paho MQTT Client and Android Service provided by Eclipse have been chosen for the implementation. According to official Paho Eclipse Github, it is possible to install the library from source or from grandle, simply adding these lines to the build.gradle of the Android Studio project:

```
dependencies {
implementation 'org.eclipse.paho:org.eclipse.paho.client.mqttv3:1.1.0'
implementation 'org.eclipse.paho:org.eclipse.paho.android.service:1.1.1'
}
```

After having specified the broker URL to which the tablet has to be connected, a name for the various topics has been chosen. In particular, as regards the publisher side, the topic has been called *Innotech/RobotCommand* while as regards the subscriber side, the topic has been called *Innotech/RobotResponse.* As already explained, the names of all the topics have to match in order to establish a proper communication since the broker provides to the subscriber only the messages associated to the topics to which it is submitted. In the *onCreate()* method all the elements designed in the .xml file associated with the activity have been declared and, after that, the *connectToBroker()* method is called which is responsible for handling all the possible cases that can occur during the attempted connections. Indeed, this method implements a try and catch structure with several ovverride methods including *onSuccess()* method and *onFailure()* method. The former, if called, displays by means of a toast that the connection has been established and set a QOS factor to 1. The latter, if called, displays by means of a toast that problems occurred that have prevented a successful connection. After having confirmed that the connection has been established, the tablet has to send the destination selected by the user to the Jetson TX2 board, as explained in the step-by-step implementation above. For this reason the *Publish()* method is called that implements a try and catch structure in which a message on *Innotech/RobotCommand* topic is sent. After that the Jetson has to send a confirmation message back to the app in order to establish a suitable communication, so the method *messageArrived()* has been implemented in such a way that it is triggered every time the broker receives a message on *Innotech/RobotResponse* topic. As it is possible to see

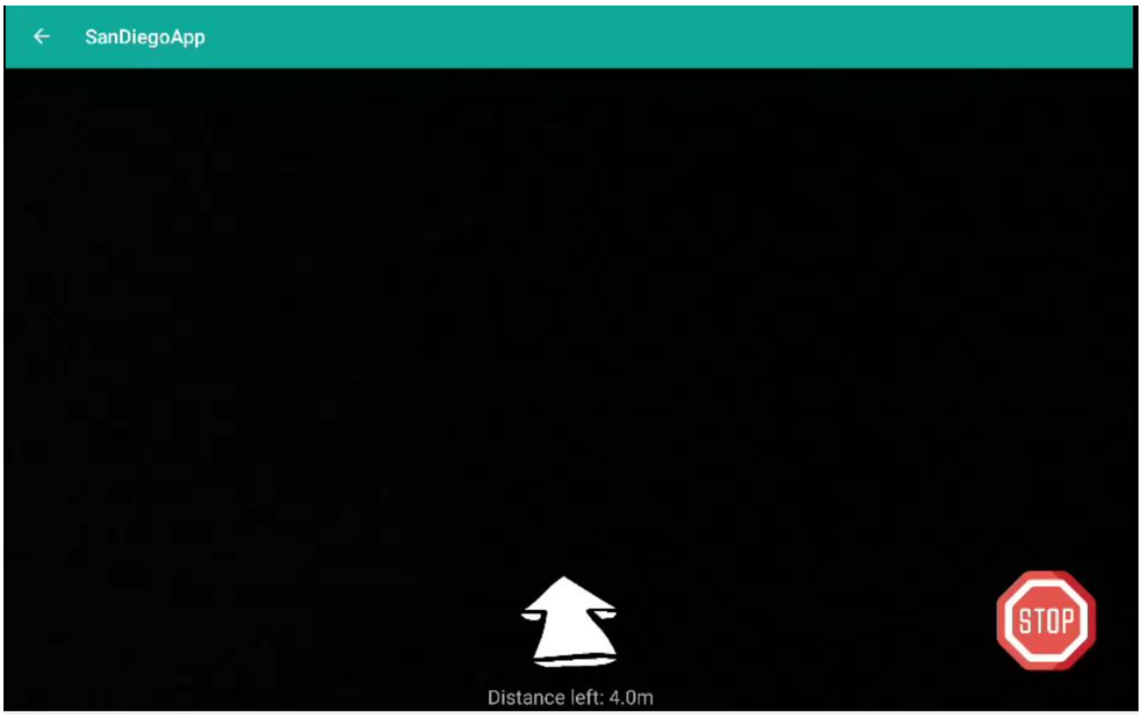


Figure 4.6: MQTT Activity

in figure 4.6, once the communication has been established, the app receives messages on *Innotech/RobotResponse* topic about the distance left and the direction to follow. These information are updated on the screen every 100ms by means of a text View and an animation Drawable represented by a blinking arrow. The stop button represents an emergency button that, once clicked, immediately stops the communication, by sending a shutdown message to the Jetson TX2 and resets the communication coming back to the homepage. Even if it is not shown on the figure 4.6, the background of the MQTT activity is filled with a texture view that streams what the rear camera of the tablet is seeing, allowing the customer to watch what the robot is seeing and being in this way more confident and sure about the path to be followed. Assuming that the communication has been established correctly and that the robot has been able to guide the passenger to the destination, the last message that has to be exchanged is the one that inform the passenger about the effective arrival to the desired destination. As soon as the Jetson board sends this message, the *SmileRate* Activity is displayed, showing a pop-up as the one in figure 4.7. In the specific case of the prototype, this technique has been adopted to obtain as many surveys as possible in order to improve the development of the software, as better explained in the next chapter. Indeed, clicking on *Rate*, a web view is created

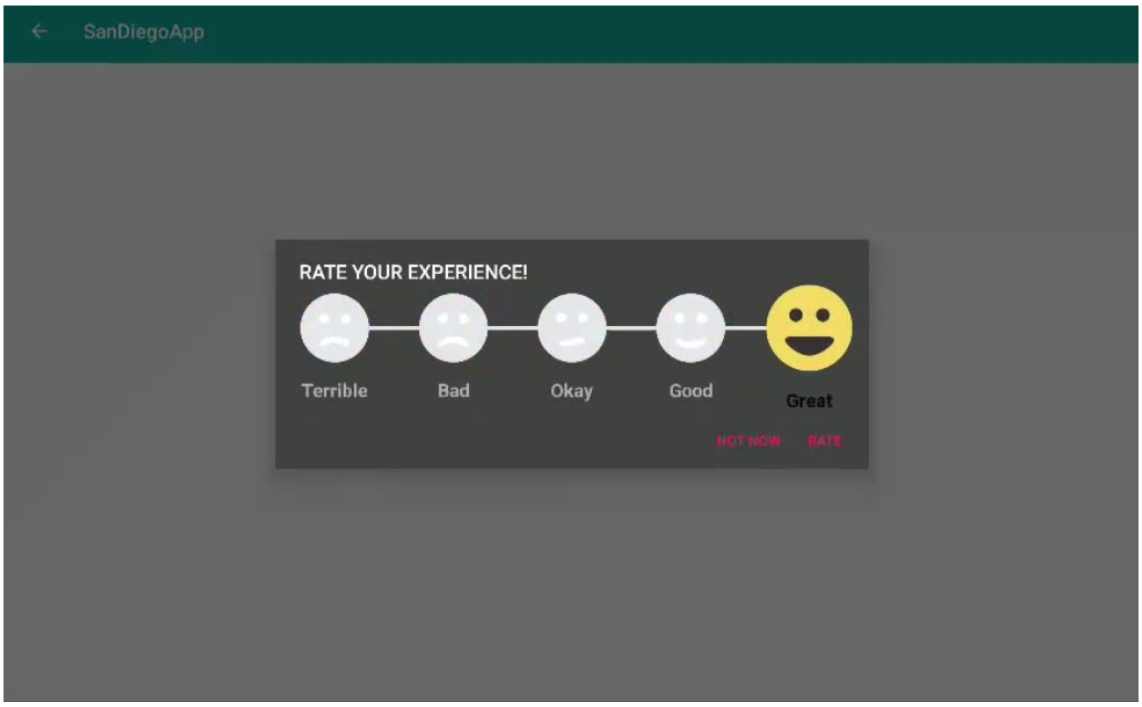


Figure 4.7: Rating PopUp

that displays an actual survey, instead of coming back to the homepage as should be expected.
In conclusion, it is possible to state that MQTT protocol represents the best compromise between a light-weight implementation and a very low communication latency since it is 20-30 times faster than a Bluetooth implementation maintaining a weight very low with the smallest packet that has only 2 bytes overhead.

4.3 Interaction with the Airport Server

As already mentioned in the previous pages, the entire communication system has also to be able to interact with the Airport facilities in order to improve the provision of the services. The original idea was to provide to the end user a series of utilities already provided by the airport but in a simplified way, by reducing the number of steps to be done to have access to them. One of the services that has been included in the Robotic User Interface was the map of Restaurants and Bars with all the useful information like opening and closure time or terminal in which they are located. The same idea has been carried out for as regards restrooms and shops. Unfortunately, the Airport decided to give the company the access to these information considered confidential, but only starting from October 2019, point at which the official tests with the actual robot have been scheduled. A similar situation has been occurred with Arrivals and Departures. In fact, as well known, every airport provides the flights information on big screens that show time, gate, airport arrival or destination, flight status, company and other useful information. The idea was to provide these information on a mobile device. In this way the information could have been available as needed to the passenger. Moreover the structure of the research has been re-designed in a different and simplified way for helping passengers to find their flight in a straightforward and more intuitive way. Even in this case, the design development has been paused waiting for the official access to the Airport database. From a series of meeting with the Airport Security district it has been discussed the modalities through which providing this access to the robot and finally it has been decided to create a JSON file where all the flights information would have been updated. For these reasons, a study on this kind of protocol has been done in the following section, focusing on the software implementation that has been made to interface the robot with the airport facilities.

4.3.1 JSON Parsing

Before entering in the implementation details of the JSON parsing on Android, it is worth to understand the importance and the general structure of the HTTP protocol, which is at the basis of the interaction between the tablet and the internet network. HTTP (Hyper Transfer Protocol), is a text language that allows the communication between client and server by means of an internet connection. As it is possible to see in figure 4.8, on the basis of the HTTP communication protocol there is the request from the client, made in a special format (the HTTP) composed by method, headers, URI, version and body.
A method is the way through which the client interacts with the resource during an HTTP request. The actions that can be performed are several, like for example *GET* that takes access to a resource from the server, *POST* that sends a resource to the server, *DELETE* that deletes a resource from the server, *PUT* that stores a resource in the server, or *HEAD* that takes access to the header of the resource,

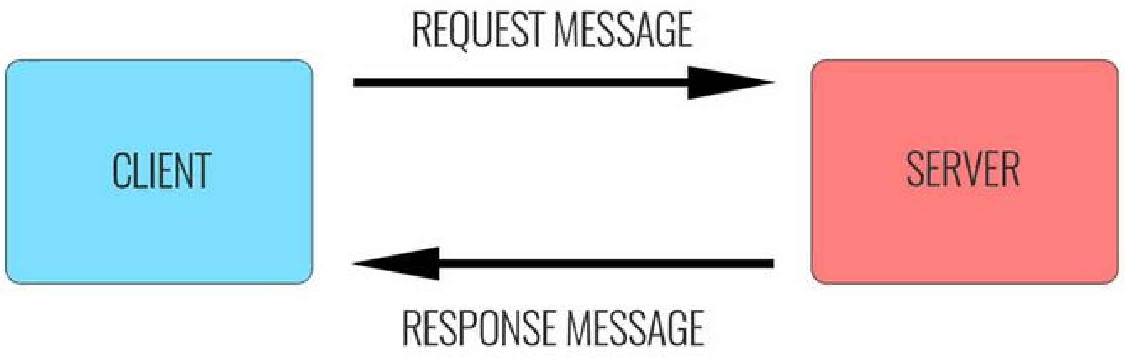
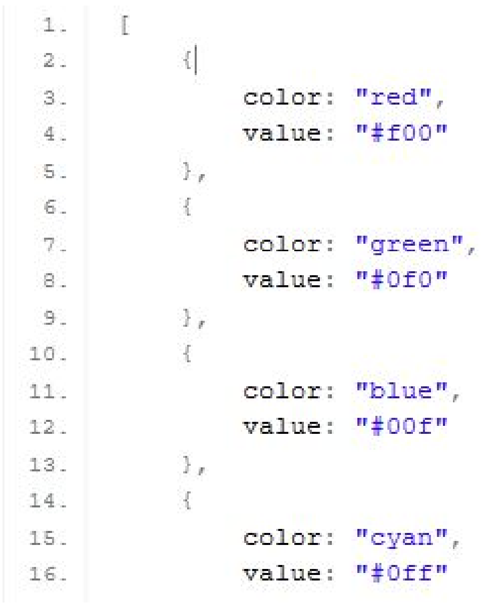


Figure 4.8: HTTP Generic Structure

ignoring the other parts.
Headers provide additional information like, as instance, about the Host or about the User-Agent, i.e. the application used by the client to carry out the request.
The URI, as the name suggest, is a Uniform Resource Identifier and has the role of making the resource unique while applying the request. There exist four options that can be made, depending on the nature of the request, but the most common is to identify a resource on an origin server, by transmitting the absolute path of the URI and the network location of it.
The version is simply the HTTP version that has been used to generate the request.
The body is used to carry the information associated with the request and its presence has to be reported by including a Transfer-Encoding header field in the request's message-headers. Once the request has been submitted following all the rules just explained, the server receives it and elaborates a suitable response. The structure of the response is equivalent to the one already analysed, with the additional information of the *Status Code*, a particular code that informs the client of the result of its request. There are several Status codes that can represent different possible operative situations: 200 means that the request has been successfully submitted, 301 that it has been moved permanently, 500 that a generic error has been encountered that did not let the communication to end well, 404 Not Found, that is one of the most common ones, means that the requested resource has not been found. The body of the response can be returned in different formats like, as instance HTML, XML or JSON. This latter, stands for *JavaScript Object Notation* and it is the most common format for data exchange especially in Android environment. Once the format has been chosen, it is the responsibility of the programmer to create a suitable interface able to make readable that format for the client.
In particular, in an Android environment this interface is managed through a library called *Volley Library* that deals with every aspect of the internet access. It is important to notice that internet connection is necessarily conditioned by the latency so it has to be handled as an asynchronous task on a secondary thread. Volley library is in charge of the correct use of this delicate resource by managing multiple

connections, caching of responses on memory and priority handling. On the basis of this library there are the concepts of *Request* and *Request Queue*. In every *Request* object there are the references to two listeners, one for the responses and one for the errors. Once the request has been made, the library takes care of deciding when it has to be executed on the basis of the Request Queue. In order to obtain the desired request from the server, a GET request has to be made to a specified URL and only if the request is successful the *onResponse()* method is executed, providing the string *result* with the content of the JSON file. Once the string result is available, the next step that has to be performed is parsing this string, i.e. analysing it and extrapolating only the desired information. A typical JSON file has the structure



```
[
    {
        color: "red",
        value: "#f00"
    },
    {
        color: "green",
        value: "#0f0"
    },
    {
        color: "blue",
        value: "#00f"
    },
    {
        color: "cyan",
        value: "#0ff"
```

Figure 4.9: JSON Generic Structure

shown in figure 4.9 where the array is included in square brackets and the single objects are included in curly brackets. The objects are separated by commas, and the inner structure is characterized by a **keyword** and a **value** associated to it. In the example in figure, *color* and *value* are the keywords while *red* and *#f00* are the values associated to the specific keyword of that specific object. Assuming that it has been requested to store the values of every object in a list, this can be done easily by means of a for cycle and with a simple string compare: the list is updated thanks to the method *.add* that, as the name suggests, adds the specific item to the list with a FIFO implementation. In a JSON file structure, the keywords are always the same, while the value associated with that keyword changes from object to object, so it is possible to store each value within the macro string *result* obtained from the original GET request.
The very same implementation structure has been done in the development of the table that had to show the flights information. In order to better understand this interaction (since it has not been possible to work with the definitive JSON file),

several tests have been conducted with external API. In particular an external API has been chosen that provided information about tennis players like their name, their country/city of origin, and an picture of them. The first parsing of the JSON file is executed in the activity where Search View is present. As already explained in Chapter 2, Search View is a normal search bar with the advantage of showing a list of query suggestions or results if available allowing the users to pick a suggestion or result to launch into. The query suggestions have been implemented through a dynamic list in which all the values of the cities have been read by parsing the JSON file. In case of the test with tennis players, cities have been substituted with players names but the software algorithm principle does not change. Once the user has picked a suggestion, the JSON file is read a second time storing the values of each object in a specific string. Thanks to this smart implementation, only the result picked from the list is displayed, reducing the sensation of confusion observed by the passengers while they looked at the flight information monitors.

Chapter 5

System Testing

Summary

This chapter will analyze all the tests that have been conducted in order to guarantee software reliability, describing step by step the procedure that has been adopted starting from single methods unit testing till the integration testing phase with both Robotic User Interface and Jetson TX2 cooperating together. After that, an analysis about the generic user testing will be carried out, explaining how the design has been developed on the basis of customer's feedback.

5.1 Software Testing

Institute of Electrical and Electronics Engineers (IEEE) defines Software Testing as: "A software process based on well-defined software quality control and testing standards, testing methods, strategy, test criteria, and tools".
Unlike a common non-I.T. product, the development of a software may depend on other systems or other software, therefore testing can be done several times during the phases of the software life cycle based on the development model adopted. Tests can be derived from requirements and specifications, design artifacts or the source code but, in general, there are different levels of testing according to the different phases of the product lifecycle [39]. As shown in figure 5.1, each phase of product development is linked to a test level:

- **Unit Testing** in which the code is decomposed in multiple functionalities implemented in units and each unit is tested independently from all the others.
- **Module Testing** that checks the software with respect to detailed design items like methods, classes, or procedures in a program.
- **Integration Testing** that checks the communication between the subsystems verifying if the pipeline is producing the expected output.

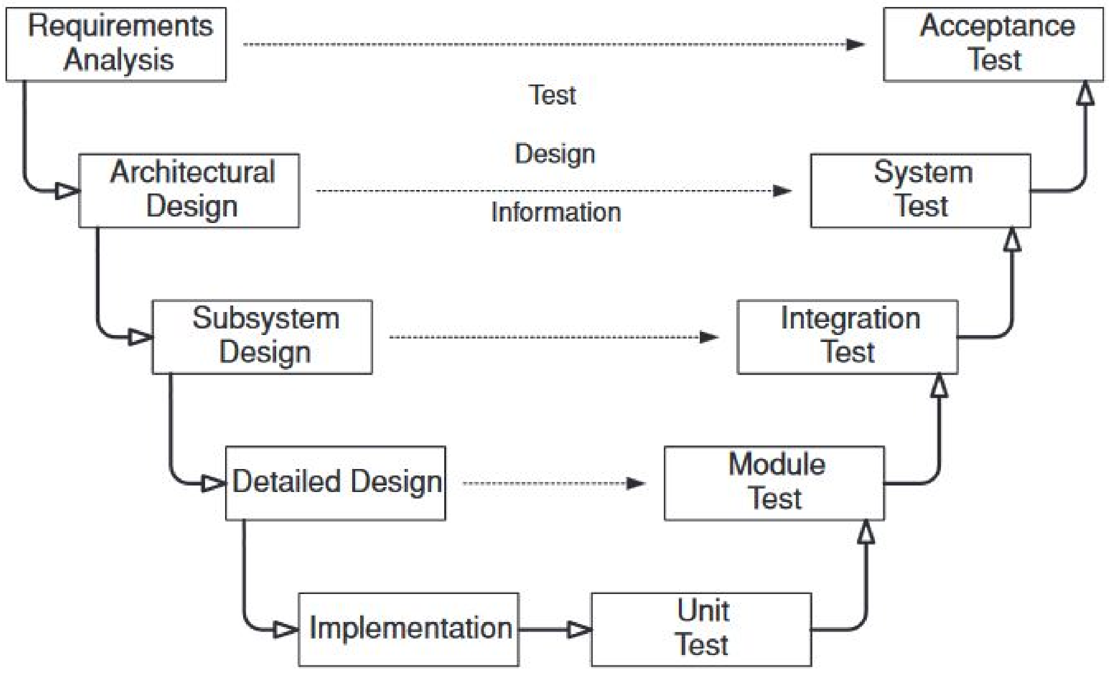


Figure 5.1: V-shape model

- **System Testing** that checks if the integrated software deployed into the final architecture leads to the expected functionalities
- **Acceptance Testing** that assesses software with respect to requirements

The model just explained is known as *V-shape Model* and is characterized by having two life cycles, one for the system and one for the software, strictly connected to each other because of the main mission of developing a product in which Hardware and Software are perfectly integrated. Because of the higher and higher number of line of codes present in an I.T. product, the time required for performing Software Testing is far above the time for the actual design and development of the product itself. During the development of the robot, attempts have been made to perform as many tests as possible on the robot prototype, considering the limited time available. These tests have been carried out in order to reduce the risk of having bugs and malfunctions on the final product and to provide to the customer a reliable product. In fact, as stated by Arthur Hicken in his article, the vast majority of bugs come in during the coding phase but the later they are found, the higher will be the cost to fix them [40]. As shown in figure 5.2, the blue line represents the percentage of bugs that occur during the different phases of software development and it can be noticed that a very high percentage of them occurs during coding phase. However, as the orange line puts on evidence, even if it is very unlikely to find defects during the coding phase, it is better to perform testing with a suitable infrastructure by creating basically a line that is the inverse of the previous one. There is then the

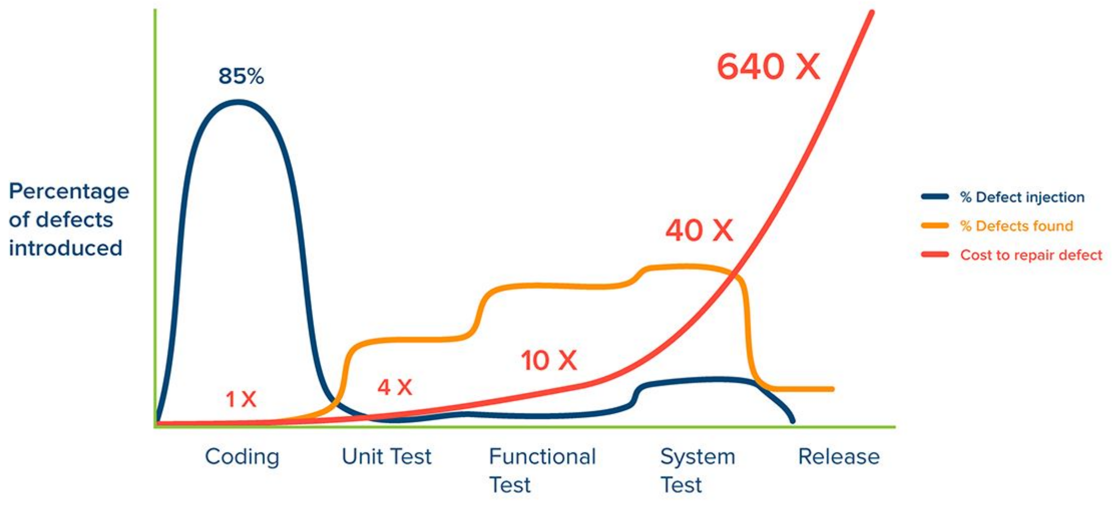


Figure 5.2: Defect Resolution Cost Timeline

last line that represents the costs to repair defects once they are found, and they increase with time extremely fast with an exponential trend. Bearing in mind these concepts and considering the limited available time, it has been decided to focus the efforts on the unit testing phase leaving the next phases aside in the expectation of the actual robot to be tested.

5.1.1 Unit testing

Unit testing represents the first step during the design development. Its main goal consists on confirming the correct operation of each unit of the design, trying to make test cases that allow to recognize possible failures. Basically an input-output relation has to be performed, verifying that everything is consistent with the specifications. In order to perform this test, the *White Box* method has been applied, that is to say: *"Deriving tests from the source code internals of the software, specifically including branches, individual conditions and statements"* [39]. This technique consists of testing a series of predefined inputs expecting a desired output, so that in case of mismatch it is evident the presence of a bug. A major White box testing technique is *Code Coverage* analysis, that eliminates gaps in the code by creating test cases that cover the untested parts of the code. There are different techniques for performing the coverage analysis:

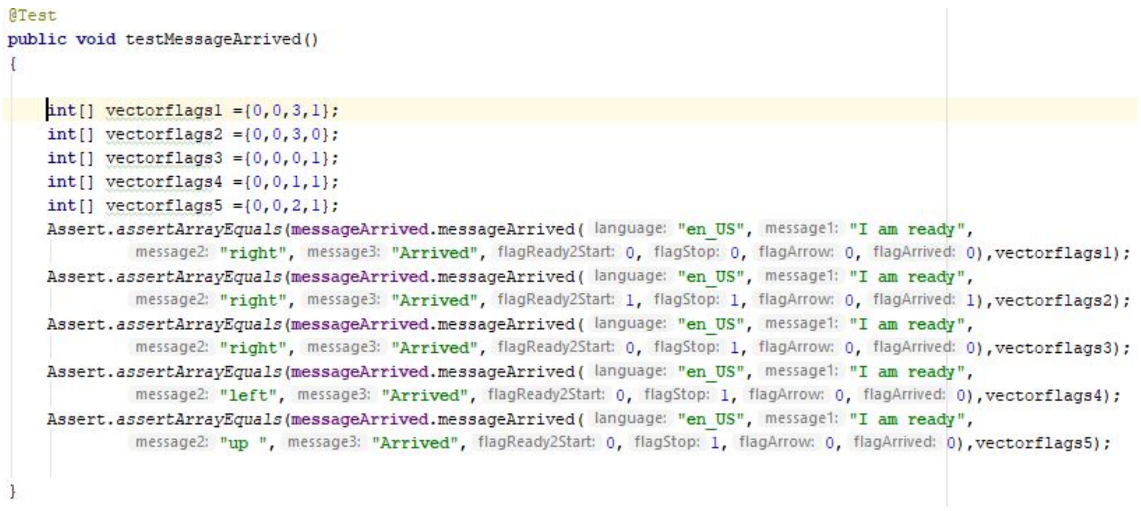
- **Statement Coverage** that covers at least once every statement present in the portion of the code under test.
- **Branch Coverage** that covers at least once every path by entering in every conditional loop present in the portion of the code under test.

Unit Testing is able to identify several bugs in the software development life cycle helping to reduce costs. However, the developed code for the only Robotic User

Interface, consists of 24 classes and more than 5000 lines of code and this made impossible to perform unit testing over all the developed code considering the available time. For this reason it has been used only on the portion of the code actually used during the tests performed at the Innovation Lab in San Diego airport, that is to say the part related to Speech Recognition and Mqtt Communication, leaving aside the portions of code not relevant for the main mission of the prototype. In order to have in mind how the Unit Testing has been performed, the *messageArrived()* method of the Mqtt Activity class will be analyzed in detail, trying to explain all the implementation phases. The first step consists of including the dependencies for performing unit testing. In particular in Android Studio IDE, it is possible to use JUnit 4, a unit testing framework for Java programming, widely used even by Android developers. In order to add JUnit 4 library in the project, a dependency has to be added to the Grandle by means of the command:

```
dependencies {
‘testImplementation ’junit:junit:4.12’
}
```

After having added the suitable dependency, two classes have been created, one called *MessageArrived* and the other called *MessageArrivedTest*. The former includes the very same *messageArrived()* method written in Mqtt class with the only difference of not being a void method but an int[ ] type. Indeed, it receives an array of input test parameters from the *MessageArrivedTest* class and return an array of integers that will be compared with the expected ones to check the absence of possible bugs. A new class has been created in order to isolate the method under test with respect to the rest of the code. The *MessageArrivedTest* class instead has been created to perform the actual test including a *testMessageArrived()* method that basically gives inputs to the *messageArrived()* method and compares the output provided by this latter with a vector containing the expected outputs. All this can be done thanks to the method *assertArrayEquals* of the JUnit library added previously. It takes as input the two arrays to compare and states that the test has been passed if and only if all the elements of the two arrays match one each other. The main idea of the entire unit testing is in fact to provide a series of known parameters through a test function and verify that the outputs match with the expected ones. As it is possible to see in figure 5.3, the method *assertArrayEquals*, given a specific set of inputs, compares desired outputs, provided by *vectorflags* array with the actual outputs provided by *messageArrived* method. In order to obtain 100% of coverage, 5 tests have been conducted changing time by time the inputs given to the *messageArrived* method. Different inputs allow to go through every conditional loop and check if there are, as instance, redundant pieces of code or particular dangerous situations in which the program gets stuck in a part of code without having the possibility to move ahead. Having a full coverage provides to have analyzed every possible operative situation with a suitable and sufficient number of tests, thus ensuring a more reliable code and a low probability of finding bugs in the following phases.



```
@Test
public void testMessageArrived()
{

    int[] vectorflags1 ={0,0,3,1};
    int[] vectorflags2 ={0,0,3,0};
    int[] vectorflags3 ={0,0,0,1};
    int[] vectorflags4 ={0,0,1,1};
    int[] vectorflags5 ={0,0,2,1};
    Assert.assertArrayEquals(messageArrived.messageArrived( language: "en_US", message1: "I am ready",
            message2: "right", message3: "Arrived", flagReady2Start: 0, flagStop: 0, flagArrow: 0, flagArrived: 0),vectorflags1);
    Assert.assertArrayEquals(messageArrived.messageArrived( language: "en_US", message1: "I am ready",
            message2: "right", message3: "Arrived", flagReady2Start: 1, flagStop: 1, flagArrow: 0, flagArrived: 1),vectorflags2);
    Assert.assertArrayEquals(messageArrived.messageArrived( language: "en_US", message1: "I am ready",
            message2: "right", message3: "Arrived", flagReady2Start: 0, flagStop: 1, flagArrow: 0, flagArrived: 0),vectorflags3);
    Assert.assertArrayEquals(messageArrived.messageArrived( language: "en_US", message1: "I am ready",
            message2: "left", message3: "Arrived", flagReady2Start: 0, flagStop: 1, flagArrow: 0, flagArrived: 0),vectorflags4);
    Assert.assertArrayEquals(messageArrived.messageArrived( language: "en_US", message1: "I am ready",
            message2: "up ", message3: "Arrived", flagReady2Start: 0, flagStop: 1, flagArrow: 0, flagArrived: 0),vectorflags5);

}
```

Figure 5.3: testMessageArrived method

5.2 Generic User Testing

Although the execution of Unit Testing with 100% coverage, the reliability of the entire code and, consequently, of the Robotic User Interface can not be considered high because not all the testing phases have been conducted. Indeed, the only unit testing does not guarantee to have a code completely bugs-free, since it checks the single components of the whole program but it does not do the same with their interaction neither among them nor with the target hardware. As already shown in figure 5.2 it can be likely to encounter bugs during the functional test or the system test where the interaction among the various methods of the classes can cause potential dangerous situations like a sudden interruption of the application.
However, as already mentioned, with the available time and resources it has been decided to perform as many tests as possible with the integrated software deployed into the final target hardware. A considerable percentage of these tests have been conducted in the presence of generic users, usually passengers of San Diego Airport, that helped not only to find and fix a good part of bugs but also to modify the structure of the interface thanks to their feedback. During the first phases of development of the interface, it has been decided to create an interface with three macro buttons, one for guiding the customer from one point to the desired destination, another one for having the robot take the customer on errands like bringing him food and the last button used for guiding the customer from one point to the desired destination, and then guiding him back. This implementation has been presented and tested but it has been figured out that it was not straightforward and created confusion because, according to users' feedback, it required too many steps to perform a specific action. Starting from these initial feedback it has been decided to reduce as much as possible the number of steps and, in particular, to implement the speech recognition algorithm that would undoubtedly help the user because of its implicit

one click implementation.
As regards speech recognition, another test has been conducted specifically asking for using it. It has been discovered that users found very convenient the use of speech recognition but at the same time they experienced difficulty in founding the microphone button through which enabling the voice assistant. For this reason a ToolTip has been implemented, that is a sort of hint for making more visible the button itself, as already explained in Chapter 2. In general these tests have been helpful not only for the development of the Robotic User Interface, but also for the development of the services that robot can provide. At the beginning of the project development, it has been decided to develop a lightweight robot with the main mission of helping passengers finding gate, restaurants and bars. However, surveys have shown that the more than 90% of the respondents admitted they considered of utmost importance not only to be helped finding the gate but also that robot offered the possibility to carry their luggage, so even the development of the mechanical design has been modified according to users' feedback.
In conclusion the possibility of running tests in an actual airport terminal gave the company the possibility to shape the development phases having a clear idea of what customers are looking for, providing a product able to give the feeling of innovation and, at the same time able to reduce airport operational costs, creating a profit for the company itself.

Chapter 6

Conclusions and Final Remarks

The final chapter is divided two main sections. The former illustrates the future enhancement as regards the development of the project in its entirety. The latter focuses on the future steps of the Robot User Interface development, central topic of this dissertation.

6.1 Robot future development

As explained in Chapter 1, the main project has been partitioned in different subsections so that each member had the possibility to study in detail a particular aspect among the several topics covered during the development of a robot. However it is worth to say that the development phases described in this thesis represent only the initial stage of a design that needs much more time and effort considering the fact that the company had an idea and started developing the robot basically from scratch. As already mentioned, the majority of design choices has been made considering two stringent requirements represented by time and budget but after having seen the progresses made in only 5 months, the airport has proposed another year of contract allowing the company to keep developing the robot with the final mission of having it on the market at the end of this additional year. Starting from September 2019, it has been decided to start running tests again but this time using the actual robot and not the prototype, so it has been decided to build the chassis and integrate an IMU and a 3D LiDAR in SLAM and obstacle detection algorithms. Thanks to the differential drive configuration the actual robot will be able to follow the path in a more accurate way as well as all the additional sensors will help the visual SLAM and obstacle avoidance. Another feature that has to be added, is the implementation of a suitable path planning algorithm that can finally allow the robot to move autonomously inside the environment without necessarily needing a pre-charged map with a predefined path to follow. From the business plan point of view, the idea that has been discussed was to rent the robot and provide software support and service, not only for the airport but also for transportation companies in general or for hospitals. Indeed, once the software has been developed, the only thing that has to change is the chassis, so the robot can be easily used in other fields that have apparently nothing in common with the airport one.

6.2 Robotic User Interface future development

After having looked through all the future steps of the robot in its entirety, this section describes the further improvements for the Robotic User Interface. The first feature to be upgraded will be the interaction with the Airport server. Indeed, after the conclusion of the contract the Airport security has been decided to authorize the sharing of those information that allow the Robotic User Interface to interact with the server by means of a JSON file. Thanks to this upgrade, features like Arrivals and Departures information and all the maps of the airport terminals can be accessible, reducing significantly airport operational costs related as instance to maintenance equipment and airport assistants. Another future enhancement consists of improving the speech recognition algorithm, by adding more and more functionalities like for example the implementation of a hands-free access to speech recognition algorithm that can increase the feeling of dealing with an innovative and reliable product. In order to obtain these goals, it has been decided to develop the software in parallel to the conduction of software testing in each phase, as already explained in Chapter 5, focusing not only on unit testing but also on functional testing and system testing, trying to develop in parallel both Hardware and Software. This mission can be achieved thanks to the additional time and budget provided by airport and its related investors. The last ambitious goal that the company has decided to follow, is to create many robots able to communicate together, collaborating on tasks in order to accomplish them in the least possible amount of time. The robots shall communicate trough MQTT protocol, sharing commands, coordinating their moves, or tell each other the actions that need to be done in order to carry out a specific task. The reason why it has been decided to follow this path is that it is much simpler to create multiple robots for multiple tasks rather than building one robot able to accomplish the task by itself. It can not be even thinkable to have a single robot able to cover the entire airport territory. Moreover it is worth to mention the economical advantage due to the reliability ensured by a group of robots since, if one has a malfunction it can be easily substituted by the others without necessarily stopping the accomplishment of the final task.

References

[1] "Hey siri: An on-device dnn-powered voice trigger for apples personal assistant." https://machinelearning.apple.com/2017/10/01/hey-siri.html. Accessed: 2019-07-16. vi, 41

[2] "Robots and robotic devices vocabulary." https://www.iso.org/obp/ui/#iso:std:iso:8373:ed-2:v1:en. Accessed: 2019-07-31. 1

[3] "Robotics investments recap: March 2019." https://www.therobotreport.com/march-2019-robotics-investments/. Accessed: 2019-06-13. 3

[4] R. G. J. Redmon, S. Divvala and A. Farhadi, "you only look once: unified, real-time object detection, university of washington, allen institute for ai, facebook ai research," 2016. 4

[5] "Yolo official website." https://pjreddie.com/darknet/yolo/. Accessed: 2019-09-30. 5

[6] L. V. Bruno Siciliano, Lorenzo Sciavicco and G. Oriolo, "Robotics, modelling, planning and control," 2009. 5

[7] K. Gorif, "Autonomous mobile robot, mechanical design," 2005. 7

[8] S. V. Robotics, "Service robotics case studies," 11 2015. 14

[9] "Relay user manual." https://static1.squarespace.com/static/53c76fe9e4b029b1ad4a55b6/t/58a382a1cd0f68dbeee92774/1487111033430/Relay_User_Manual.pdf. Accessed: 2019-06-14. 15

[10] "Tug t3 autonomous mobile robot datasheet." https://aethon.com/PDF/T3_Datasheet.pdf. Accessed: 2019-06-15. 15

[11] "Machine-as-a-service 24/7 autonomous operation." https://www.knightscope.com/knightscope-k3. Accessed: 2019-06-15. 16

[12] "Lg expands iot ecosystem with lineup of futuristic robotic products." https://www.lg.com/us/PDF/press-release/CES2017_LG_Robot_Lineup_Release_FINAL_01_04_17.pdf. Accessed: 2019-06-15. 16

[13] C. Prof. Akram Hossain, Purdue University, "Hmi design: An analysis of a good display for seamless integration between user understanding and automatic controls," 2012. 20

[14] N. K. Karl F. MacDorman, "The uncanny valley," 2012. 20

[15] "Mobile operating system market share worldwide." `http://gs.statcounter.com/os-market-share/mobile/worldwide`. Accessed: 2019-06-21. 24

[16] "Platform architecture." `https://developer.android.com/guide/platform`. Accessed: 2019-06-21. 24

[17] C. Aliferi, "Android programming cookbook," 2016. 26

[18] "Activity lifecycle." `https://developer.android.com/guide/components/activities/activity-lifecycle.html`. Accessed: 2019-06-24. 27

[19] I. G. Clifton, "Android user interface design, implementing material design for developers," 2015. 31

[20] "Material design." `https://material.io/`. Accessed: 2019-06-18. 32, 34, 36

[21] H. Z. Jianliang Meng, Junwei Zhang, "Overview of the speech recognition technology," 2012. 39

[22] C. B. F. B. Yu Zhong, T.V. Raman and J. P. Bigham, "Justspeak: Enabling universal voice control on android," 2014. 39

[23] S. Reehal, "Siri the intelligent personal assistant," 2016. 40

[24] "Voice command detection." `https://docs.nvidia.com/isaac/isaac/packages/audio/doc/voice_command_detection.html`. Accessed: 2019-07-17. 42, 43

[25] L. R.Rabiner and R. W. Schafer, "Theory and applications of digital speech processing," pp. 463–465, 2011. 42

[26] "Tensorflow lite inference." `https://www.tensorflow.org/lite/guide/inference`. Accessed: 2019-07-18. 43

[27] "5 differences between voice and speech recognition." `https://biometrictoday.com/5-differences-between-voice-and-speech-recognition/`. Accessed: 2019-07-19. 45

[28] L. R.Rabiner, "A tutorial on hidden markov models and selected applications in speech recognition," pp. 257–286, 1989. 46, 48, 50, 54, 55, 56

[29] D. Jurafsky and J. H. Martin, "Speech and language processing," pp. 464–558, 2018. 47, 48, 49, 51, 52, 53

[30] R. S. Chavan and G. S. Sable, “An overview of speech recognition using hmm,” pp. 235–236, 2013. 55

[31] S. Primorac and M. Russo, “Android application for sending sms messages with speech recognition interface,” 2012. 55

[32] “Speech recognizer.” `https://developer.android.com/reference/android/speech/SpeechRecognizer.html`. Accessed: 2019-07-26. 56

[33] D. W. Gage, “Network protocols for mobile robot systems,” October 1997. 60

[34] F. S. Dennis Krupke, Jianwei Zhang, “Impact: A holistic framework for mixed reality robotic user interface classification and design,” April 2019. 60

[35] M. Mock and E. Nett, “Real-time communication in autonomous robot systems,” December 1998. 61, 63

[36] I. Puy, “Bluetooth,” 2008. 68

[37] “What is mqtt?.” `https://mqtt.org/faq`. Accessed: 2019-07-02. 69

[38] “Mqtt essentials.” `https://www.hivemq.com/mqtt-essentials/`. Accessed: 2019-07-02. 69

[39] J. O. Paul Ammann, “Introduction to software testing,” 2008. 80, 82

[40] A. Hicken, “The shift-left approach to software testing,” 2018. 81

[41] R. Murphy, “Usb 101: An introduction to universal serial bus 2.0,” 2017.