University of Western Macedonia

FACULTY OF ENGINEERING

Department of Informatics & Telecommunications Engineering

Laboratory of Digital Systems and Computer Architecture

Secure Autonomous Cloud Brained Humanoid Robot for Search and Rescue missions in Hazardous Environments

---

Diploma Thesis

Georgios Angelopoulos

Supervisor : Dr. Minas Dasygenis

October 22, 2018

Kozani

Declaration of Authorship

Δηλώνω ρητά ότι, σύμφωνα με το άρθρο 8 του Ν. 1599/1986 και τα άρθρα 2,4,6 παρ. 3 του Ν. 1256/1982, η παρούσα Διπλωματική Εργασία με τίτλο "Secure Autonomous Cloud Brained Humanoid Robot for assisting rescuers in Hazardous Environments" καθώς και τα ηλεκτρονικά αρχεία και πηγαίοι κώδικες που αναπτύχθηκαν ή τροποποιήθηκαν στα πλαίσια αυτής της εργασίας και αναφέρονται ρητώς μέσα στο κείμενο που συνοδεύουν, και η οποία έχει εκπονηθεί στο Τμήμα Μηχανικών Πληροφορικής και Τηλεπικοινωνιών του Πανεπιστημίου Δυτικής Μακεδονίας, υπό την επίβλεψη του μέλους του Τμήματος κ. Μηνά Δασυγένη αποτελεί αποκλειστικά προϊόν προσωπικής εργασίας και δεν προσβάλλει κάθε μορφής πνευματικά δικαιώματα τρίτων και δεν είναι προϊόν μερικής ή ολικής αντιγραφής, οι πηγές δε που χρησιμοποιήθηκαν περιορίζονται στις βιβλιογραφικές αναφορές και μόνον. Τα σημεία όπου έχω χρησιμοποιήσει ιδέες, κείμενο, αρχεία ή / και πηγές άλλων συγγραφέων, αναφέρονται ευδιάκριτα στο κείμενο με την κατάλληλη παραπομπή και η σχετική αναφορά περιλαμβάνεται στο τμήμα των βιβλιογραφικών αναφορών με πλήρη περιγραφή. Απαγορεύεται η αντιγραφή, αποθήκευση και διανομή της παρούσας εργασίας, εξ ολοκλήρου ή τμήματος αυτής, για εμπορικό σκοπό. Επιτρέπεται η ανατύπωση, αποθήκευση και διανομή για σκοπό μη κερδοσκοπικό, εκπαιδευτικής ή ερευνητικής φύσης, υπό την προϋπόθεση να αναφέρεται η πηγή προέλευσης και να διατηρείται το παρόν μήνυμα. Ερωτήματα που αφορούν τη χρήση της εργασίας για κερδοσκοπικό σκοπό πρέπει να απευθύνονται προς τον συγγραφέα. Οι απόψεις και τα συμπεράσματα που περιέχονται σε αυτό το έγγραφο εκφράζουν τον συγγραφέα και μόνο.

Acknowledgments

I would like to thank my supervisor, Dr. Minas Dasygenis, Professor at the University of Western Macedonia, for the patient guidance, encouragement, and advice he has provided throughout my time as his student. A very special gratitude goes out to my close friends and fellow students for their support and friendship. For working together and building the right mindset to achieve higher goals in life and science. Finally, I must express my very profound gratitude to my parents for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of writing this thesis. This accomplishment would not have been possible without them. Thank you.

Abstract

Cloud robotics is a field of robotics that attempts to invoke cloud technologies centered on the benefits of converged infrastructure and shared services for robotics. When connected to the cloud, robots can benefit from the powerful computation, storage, and communication resources of a modern data center in the cloud.

Nowadays there have been a few attempts that combine cloud with humanoid robots but they haven't been used as a service for emergency response in hazardous environments. These cloud-based applications can get slow due to high latency responses and can be hacked very easily.

This thesis proposes a novel, fast and secure system using a cloud brained humanoid robot programmable to be able to autonomously move and detect lives in emergency scenarios with the potential to communicate with the victim. Finally, to explore the safety of our system, we performed an experiment where a penetration tester tries to hijack our robot.

Keywords: Humanoid Robot, Cloud-Brained, Cloud Robotics, Human-Robot Interaction

Abstract in Greek

Η ρομποτική νέφους είναι ένας κλάδος της ρομποτικής που επικαλείται τεχνολογίες υπολογιστικού συννέφου με επίκεντρο τα πλεονεκτήματα της σύγκλισης υποδομών και κοινών υπηρεσιών για τη ρομποτική. Όταν συνδέονται στο νέφος, τα ρομπότ μπορούν να επωφεληθούν από τους ισχυρούς υπολογιστές, αποθηκευτικούς και επικοινωνιακούς πόρους ενός σύγχρονου κέντρου δεδομένων.

Σήμερα έχουν γίνει μερικές προσπάθειες που συνδυάζουν το υπολογιστικό νέφος με την ρομποτική και τα ανθρωποειδείς ρομπότ, αλλά δεν έχουν χρησιμοποιηθεί ως υπηρεσία αντιμετώπισης έκτακτων περιστατικών σε επικίνδυνα περιβάλλοντα. Αυτές οι εφαρμογές που βασίζονται στην συγκεκριμένη τεχνολογία μπορεί να είναι αργές λόγω υψηλής καθυστέρησης στην επικοινωνία και μπορούν να παραβιαστόυν πολύ εύκολα.

Η συγκεκριμένη διπλωματική εργασία προτείνει ένα νέο, γρήγορο και ασφαλές σύστημα χρησιμοποιώντας ένα ανθρωποειδές ρομπότ που έχει προγραμματιστεί για να μπορ-εί να μετακινείται αυτόνομα και να ανιχνεύει ζωές σε σενάρια έκτακτης ανάγκης ενώ έχει την δυνατότητα να επικοινωνεί με το θύμα. Τέλος, για να διερευνήσουμε την ασφάλεια του συστήματός μας, πραγματοποιήσαμε ένα πείραμα όπου ένας δοκιμαστής διείσδ-υσης προσπαθεί να παραβιάσει το ρομπότ μας.

Λέξεις κλειδιά: Ανθρωπόμορφο ρομπότ, Υπολογιστικό Νέφος, Ρομποτική του Συννέφου, Αλληλεπίδραση Ανθρώπου-ρομπότ

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction and Background

1.1 Introduction

1.1.1 Cloud Robotics

In recent years, research in robotics has become one of the most popular research fields in industry and academia, also, with the recent popularity of cloud computing [1], there has been an interest in applying similar concepts to robotics and autonomous systems. Cloud robotics [2] allows robots to take advantage of the rapid increase in data transfer rates to offload tasks without hard real-time requirements. This is of particular interest for mobile robots where onboard computation entails additional power requirements which may reduce operating duration and constrain robot mobility as well as increased costs. Cloud-Based robots have proved to have several applications and advantages over the traditional networked based robots.

They have the ability to offload computation-intensive tasks to the cloud. The robots only have to keep necessary sensors, actuators, and basic processing power to enable real-time actions (e.g., real-time control). It extends the battery life, and the robotic platform becomes lighter and less expensive with easier to maintain hardware. The maintenance of software onboard with the robots also becomes simpler, with less need for regular updates. As the cloud hardware can be upgraded independently from the robotic network, the operational life and usefulness of the robotic network can be easily extended.

The robots can gain information and knowledge to execute tasks through databases in the cloud. They do not have to deal with the creation and maintenance of such data.

The cloud provides a medium for robots to share information and learn new skills and knowledge from each other. The cloud can host a database or library of skills or behaviors that map to different task requirements and environmental complexities.

Several research groups have started to explore the use of cloud technologies in robotic applications. For example, self-driving cars in Google are one type of cloud-connected robot. The autonomous cars access data from Google Maps and images stored in the cloud to recognize their surroundings. They also gather information about road and traffic conditions and send that information back to the cloud. A research group at Singapore's ASORO laboratory has built a cloud computing infrastructure to generate 3-D models of environments, allowing robots to perform simultaneous localization and mapping (SLAM) much faster than by relying on their onboard computers [3]. At LAAS, Florent Lamiraux, Jean-Paul Laumond, et al. are creating object databases for robots to simplify the planning of manipulation tasks like opening a door. The idea is to develop a software framework where objects come with a "user manual" for the robot to manipulate them. This manual would specify, for example, the position from which the robot should manipulate the object. The approach tries to break down the computational complexity of manipulation tasks into simpler, decoupled parts: a simplified manipulation problem based on the object's "user manual," and a whole-body motion generation by an inverse kinematics solver, which the robot's computer can solve in real time.

The current cloud robotics applications have many advantages as mentioned but also have drawbacks as well. Sometimes there are difficulties in controlling robots motion as it depends on sensors, feedback of controllers and internet connections. Apart from that, many times cloud-based applications can get slow due to high latency responses or a network problem. Last but not least, it can be hacked easily as it requires real-time execution for the same.

1.1.2 Robots in Hazardous environments

The DOE (U.S. Department of Energy) Environmental Restoration and Waste Management Robotics Technology Development Program explains that manual work within hazardous environments is slow and expensive [4]. Worker efficiency is low due to protective clothing and, in some cases, exposure limits that require work to be

accomplished in several minute intervals. Even when exposure limits are not an issue, fatigue is often induced by the confined spaces and by the highly repetitive nature of certain tasks. The cost of a given project is increased because of the special materials needed to protect workers and the environment, and because of the additional wastes generated in the form of contaminated clothing, rags, tools, etc.. Moreover, the time required to accomplish missions in a hazardous environment is adversely impacted not only by low worker efficiency but also by the need to prepare the workers and instrument the site. For these reasons, the use of robots in hazardous environments is crucial in terms of occupational safety of workers and the health of rescue and other operations.

Some robots, such as "PackBot" developed by iRobot [5], "Quince" developed by Chiba Institute of Technology collaboration with Tohoku University and International Rescue System [6], and "Survey Runner" developed by TOPY Industries Ltd. [7], are deployed in the Fukushima Daiichi Nuclear Power Plant. These robots are crawler type robots and are controlled by the operator who remains in a safe area. They measure temperature and radioactivity in hazardous zones and also transmit video and sensory information back to the operator. They are performing duties that are very important at the beginning of restoration, but there are several tasks that are hard for them to execute.

In contrast with other authors, we have developed a secure autonomous cloud brained humanoid Robot for search and rescue operations with the ability to communicate with the victim. Our robot has the ability to recognize humans and immediately alert rescuers. We also have developed a secure web application as a control panel for the rescuers. Our application contains information about the victim (a photo captured from the robot) but also the real-time position of our humanoid robot. Our humanoid robot is a key factor for providing emergency psychological aid to the injured in an emergency area.

1.1.3 Humans and Robots Interaction

There has been some research on human interactions with a humanoid robot [8]. Related research [9] on using robots in autism research mainly focused on behaviors that increase and maintain children's engagement in interacting with the robots, such

as eye-to-eye gaze. Engagement is not only an issue for autistic children but also for public interest in STEM education.

Investigating the reactions of children and adults to a robot dog at a shopping mall [10] showed that children developed positive emotions toward the robot dog at the visceral level, at the behavioral level, and at the reflective level. The children became excited when they first saw the robot dog, then they played with the robot dog, and they expressed the wish to bring the dog home. However, a robot dog has fewer potential functions than a humanoid robot and plays a significantly different role from a humanoid robot. It is expected that children's interactions with a humanoid robot would differ from that with a robot dog but similarly to how they interact with other humans.

Studies have shown [11], [12] that trust towards automation affects reliance (i.e. people tend to rely on automation they trust and not use automation they do not trust). For example, trust has frequently been cited [13], [14] as a contributor to human decisions about monitoring and using automation. Indeed, within the literature on trust in automation, complacency is conceptualized interchangeably as the overuse of automation, the failure to monitor automation, and lack of vigilance. For optimal performance of a human-automation system, human trust in automation should be well-calibrated.

In [15], trust is conceived to be an "attitude that an agent (automation or another person) will help achieve an individual's goals in a situation characterized by uncertainty and vulnerability." A majority of research in trust in automation has focused on the relationship between automation reliability and operator usage often without measuring the intervening variable, trust. The utility of introducing an intervening variable between automation performance and operator usage, however, lies in the ability to make more precise or accurate predictions with the intervening variable than without it. This requires that trust in automation be influenced by factors in addition to automation reliability/performance. The three dimensional (Purpose, Process, Performance) model proposed by Lee and See [15], for example, presumes that trust (and indirectly, propensity to use) is influenced by a person's knowledge of what the automation is supposed to do (purpose), how it functions (process), and its actual performance. While such models seem plausible, support for the contribution

of factors other than performance has typically been limited to the correlation between questionnaire responses and automation use. Despite multiple studies in trust in automation, the conceptualization of trust and how it can be reliably modeled and measured is still a challenging problem.

In contrast to automation where system behavior has been pre-programmed and the system performance is limited to the specific actions it has been designed to perform, autonomous systems/robots have been defined as having intelligence based capabilities that would allow them to have a degree of self-governance, which enables them to respond to situations that were not pre-programmed or anticipated in the design. Therefore, the role of trust in interactions between humans and robots are more complex and difficult to understand.

1.2 Background

I was one of the three researchers who during the year 2017 worked on creating an Internet of Things Humanoid Robot Teleoperated by an Open Source Android Application [16]. Our paper proposed a novel open source platform Android application for controlling a humanoid robot, with more features than other research projects, in remote areas either behind NAT (Network Address Translation) or without it. The user could view what the robot is seeing through the Android application and could also move it. Additionally, our system had the ability to deliver real-time audio through our application.

However, the initial system was quite slow, which meant that the telemanipulation did not feel natural for the person controlling the robot. As a result, I continued working on the project during my diploma thesis. The focus of my diploma thesis was to identify the bottlenecks in the initial system when it comes to communication between the bridge server but also to add decisive elements for making it an attractive solution in search and rescue missions.

1.3 Problem Description

The main goal for this thesis is to create a fast and secure system using a cloud brained humanoid robot programmable to be able to autonomously move and detect

lives in emergency scenarios with the potential to communicate with the victim. To make our humanoid robot more effective, it is important to have a system with low latency but also a secure encrypted system due to transferring sensitive information.

In this diploma thesis, the communication between the server and the humanoid robot was attempted using socket programming. The goal is to use the server as an operational decision center which sends directly to the robot the essential commands. Part of the focus of the thesis will be to evaluate if our proposal is a suitable tool for assisting rescuers in hazardous environments searching for lives.

1.4 Equipment

1.4.1 Humanoid Robot

Currently, the cost of a humanoid robot is dissuasive for labs or research teams that cannot build reliable legged robots. Functional robots must be devised by researchers and engineers, and specialized companies must manufacture them to ensure good industrial integration.

Among performant robots, the Asimo humanoid built by Honda may be the most impressive [17], [18]. It is capable of walking fast, up to 3 km/h forward, change direction and walk up/down stairs smoothly. It can even reach 7 km/h by adopting a special running gait. It can also react to external disturbances by adjusting its posture to keep stability.

The HRP-2 robot manufactured by Kawada Industries is also a good technological achievement [19], [20], and [21]. It can walk up to 2.5 km/h and can lie down and get up again by itself. Their successors [22] should help people in their everyday life or achieve tedious tasks in the place of humans. The size of these robots must be compatible with human-scale environments.

However, these robots were either not available to researchers or only available to the few teams that have enough funding to support the cost and maintenance of such robots. Functional humanoids are somewhat expensive (see Table 1.1). Even the HOAP small-sized humanoid robot from Fujitsu costs about 50K US $. The NAO robot has been devised with the concern of cost reduction without sacrificing quality and performance. It is a completely custom designed robot as the whole

process of design and manufacturing is mastered (mechanics, electronics, software). This allowed costs to be reduced at every stage of design. The company employs subcontractors to produce plastic parts or electronic circuits on a large scale. One way to achieve cost reduction was the reuse of the same actuator modules for several joints. Another way comprised reducing the number of motors without sacrificing mobility. The robot cost about 10K euros for laboratories. Thanks to mass production and reduction of functionalities a version will be publicly available for approximately 5K euros.

Table 1.1: Characteristics of Functional Humanoids.

	Height(m)	**Weight(kg)**	**BMI(kg/m2)**	**Price**
KHR-2HV	0.34	1.3	10.9	1K US $
HOAP	0.50	7.0	28.0	50K US $
NAO	**0.58**	**4.5**	**13.5**	**10K €**
QRIO	0.58	6.5	19.0	NA
ASIMO	1.30	54.0	32.0	NA
REEM-A	1.40	40.0	20.4	NA
HRP-2	1.54	58.0	24.5	400K US $
Human	1.5-2	50-100	18-25	NA

BMI: Body mass index NA: not available

Therefore, the experiments were performed on an Aldebaran NAO humanoid robot, version 5 (Figure 1.1). The robot is 573.2 mm high and 273.3 mm wide. From scapula to the end of the hand, the length of the arms is 290 mm. Detailed information about the geometry of the NAO can be found in Figure 1.2.

Nao has 2 ultrasound devices situated in the chest that provides space information in 1-meter range distance if an object is situated at 30 degrees from the robot chest (60 degrees all cone combining both devices).

Also has a bumper which is a contact sensor that help us know if the robot is touching something, in this case, the bumpers are situated in front of each foot and they can be used, for example, to know if the robot is kicking the ball or if there are some obstacles touching our feet.

Nao has 8 Force Sensing Resistors (FSR) situated at the sole of feet. There a 4 FSRs in each foot, then. The value returned for each FSR is a time needed by a capacitor to charge depending on the FSR resistor value. It is not linear (1/X) and needs to be calibrated. When no force is applied the sensor reading is 3000 and when the sensor reading is 200 means that is holding about 3 kg. These sensors are

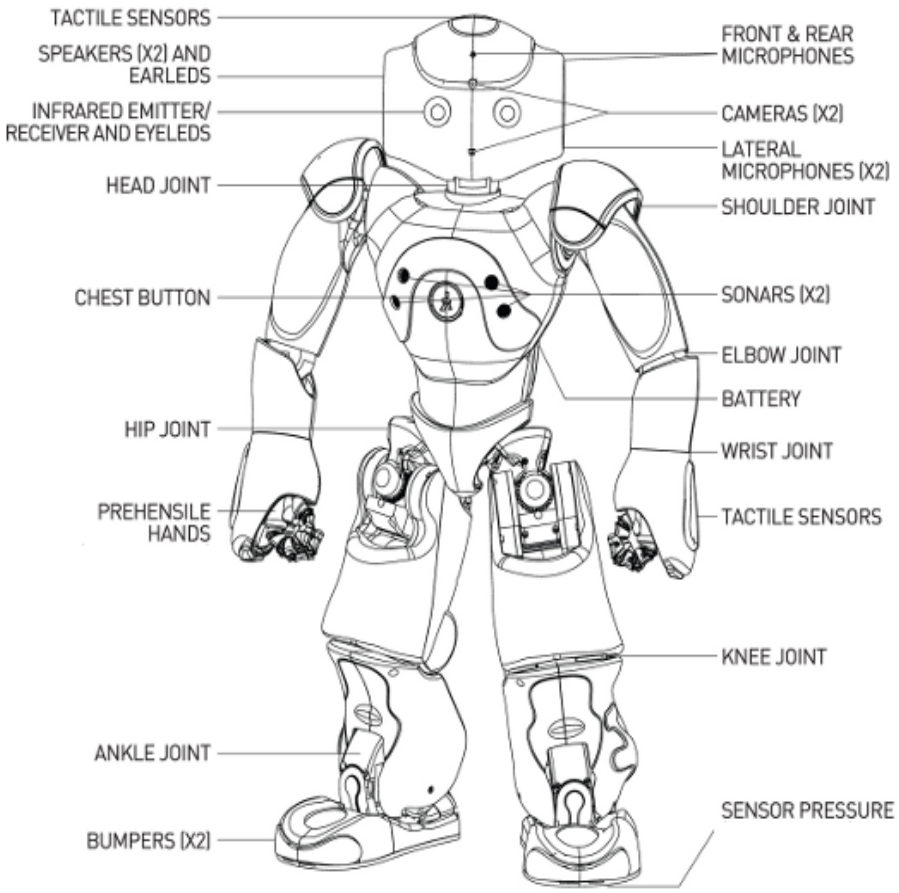


Figure 1.1: NAO version 5. (Image from Aldebaran's documentation)

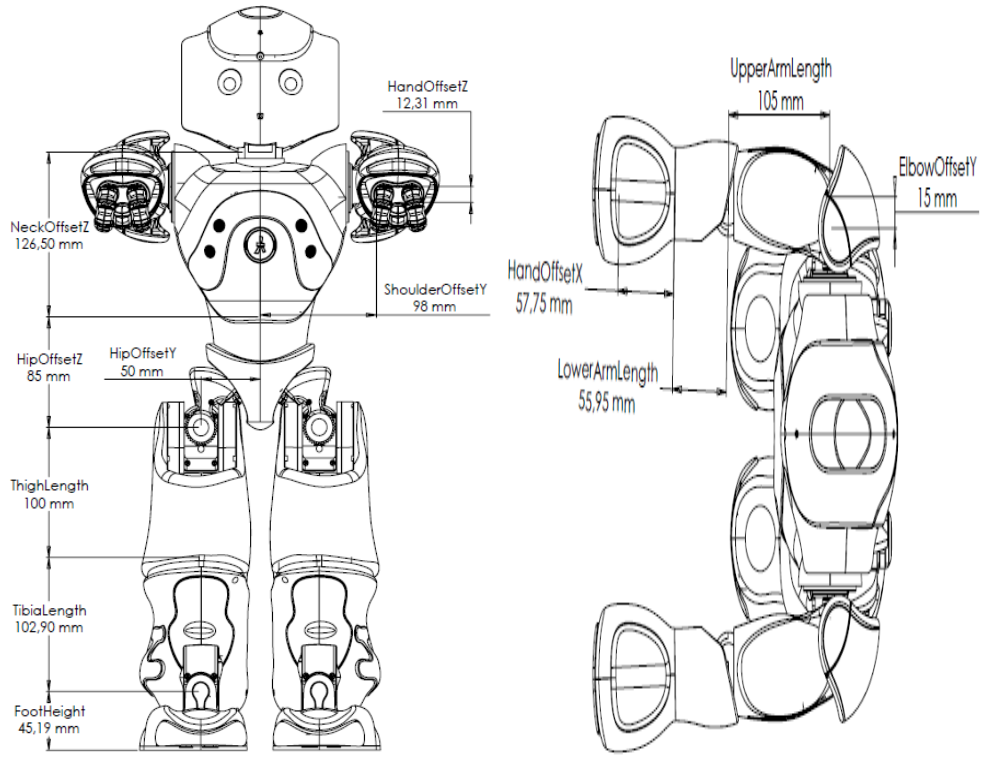
useful when we are generating movements sequences to know if one position is a zero moment pose (ZMP) and its sensors can be complemented with inertial sensors.

Nao has a gyrometer and an accelerometer. These sensors are two important devices when we are talking about motion concisely Kinematics and Dynamics. These sensors help us to know if the robot is in a stable position or in an unstable one when the robot is walking

The placement of the two cameras, which our humanoid is equipped with, is such that they do not overlap. There is a gap of 5.2 degrees between the two fields of view. The first camera is facing straight forward and the second camera faces down 39.7 degrees. See Figure 1.3 for more details.

1.4.2 Server

The NAO is equipped with a 500 MHz x86 processor from Advanced Micro Devices Inc. Because of the limited processing capacity of this processor, all data processing was performed using a cloud server.



(a) Front View (b) Top View

Figure 1.2: Geometry of the NAO. (Image from Aldebaran's documentation)

The server we used was an Ubuntu 16.04.4 LTS web server with 4096 MB RAM and 2 CPU cores. The weekly CPU utilization can be found in Figure 1.4.

Our server is on Okeanos [23]. Okeanos is GRNET's cloud service for the Greek Research and Academic Community, providing cloud services such as virtual machines, networks, and storage. It is powered by Synnefo, a complete open source cloud stack written in Python that provides Compute, Network, Image, Storage services. Synnefo manages multiple Ganeti clusters at the backend and uses the OpenStack API at the frontend.

The only user-visible Okeanos services are Cyclades and Pithos. Cyclades is the Compute and Network part of Okeanos. It provides access to the virtual machines that can be created, booted, shutdown or destroyed on demand, and to networking functionalities including firewalls, Internet access, and virtual networks. Cyclades also keeps the statistics of the VMs concerning the compute and network resources that are used.

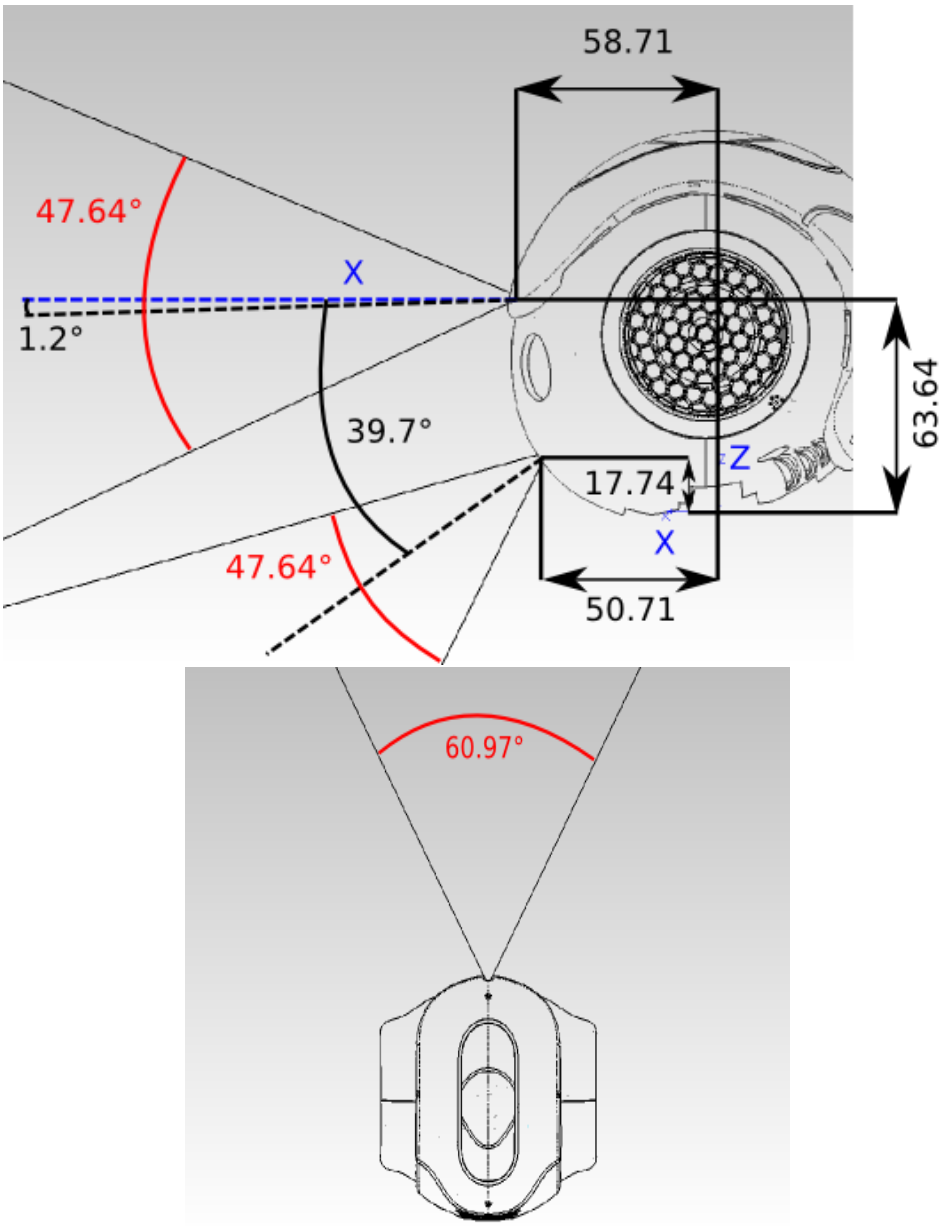


Figure 1.3: Location of the Video Camera. (Image from Aldebaran's documentation)

1.5 The Following work structure

The rest of this thesis is structured as follows.

Chapter 2 will start by discussing the tools that help us in order to create the code of our system but also the tools we needed in order to connect to our cloud server. Afterward, we will continue by introducing the choice of programming languages and software for this thesis. Some of the choices regarding the socket programming and the encryption for the communication will be explained. Chapter 2 ends with some necessary theory related to OpenCV and image processing.

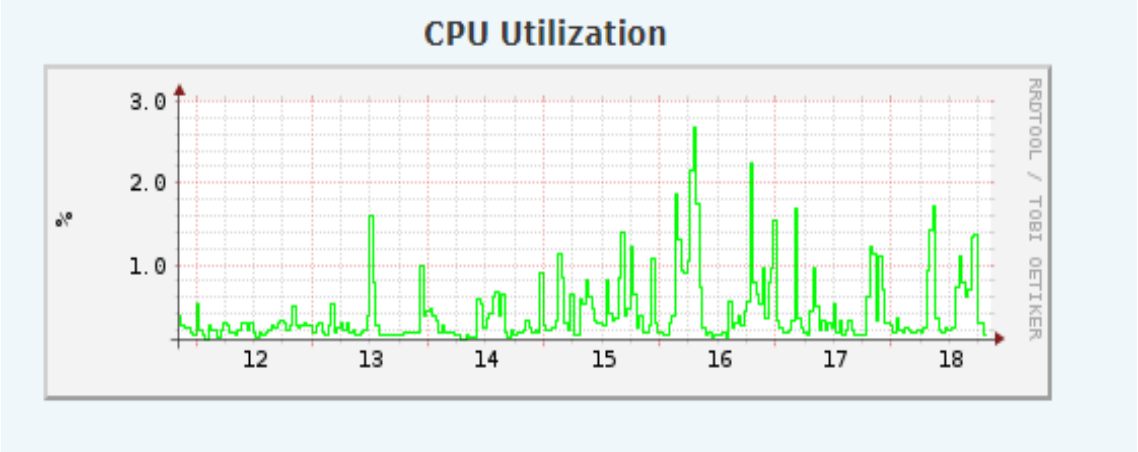


Figure 1.4: CPU Utilization.

In chapter 3 we will talk about our system architecture some choices regarding movement for the robot will be explained. A large part of chapter 4 is devoted to obstacle detection. Also, we will talk about important functions that our robot uses. Finally, we will present how the cloud server and the website works.

At the beginning of chapter 4, we are taking a closer look at how and if our humanoid robot can work on extreme conditions. Apart from that, we discuss the experiment we performed where a penetration tester tries to hijack our robot. Last but not least, we present our SWOT analysis.

In chapter 5, the conclusions of this thesis is presented. The results and suggestions for further work are included.

Chapter 2

Background Theory

2.1 Tools

In this section, we will discuss the tools that help us connect to our server. Afterward, we will talk about the tools we will need in order to create the code of our system.

2.1.1 Winscp

WinSCP (Windows Secure Copy) is a free and open-source SFTP, FTP, WebDAV, Amazon S3 and SCP client for Microsoft Windows. Its main function is secure file transfer between a local and a remote computer. Beyond this, WinSCP offers basic file manager and file synchronization functionality. For secure transfers, it uses Secure Shell (SSH) and supports the SCP protocol in addition to SFTP.

Development of WinSCP started around March 2000 and continues. Originally it was hosted by the University of Economics in Prague, where its author worked at the time. Since July 16, 2003, it is licensed under the GNU GPL and hosted on SourceForge.net.

2.1.2 Putty

PuTTY is a free and open-source terminal emulator, serial console and network file transfer application. It supports several network protocols, including SCP, SSH, Telnet, rlogin, and raw socket connection. It can also connect to a serial port. The name ”PuTTY” has no official meaning.

PuTTY was originally written for Microsoft Windows, but it has been ported to various other operating systems. Official ports are available for some Unix-like

platforms, with work-in-progress ports to Classic Mac OS and macOS, and unofficial ports have been contributed to platforms such as Symbian, Windows Mobile and Windows Phone.

PuTTY was written and is maintained primarily by Simon Tatham.

2.1.3 GNU nano

GNU nano is a text editor for Unix-like computing systems or operating environments using a command line interface. It emulates the Pico text editor, part of the Pine email client, and also provides additional functionality. Unlike Pico, nano is licensed under the GNU General Public License (GPL). Released as free software by Chris Allegretta in 1999, nano became part of the GNU Project in 2001.

2.2 Programming language

During the work on my research project in 2017 [16], it became clear that it was more beneficial to control the robot by writing scripts in a standard programming language than to use the Choreographe-program which came with the robot. The main issue was that the block-based control methods in Choreographe were thought not to be flexible enough. When wanting to write scripts in a standard programming language, it was necessary to find the language most suitable for the desired functionality of the system. The operating system that runs on the NAO robot is called NAOqi. Aldebaran Robotics has also released a Software Development Kit (SDK) to all users disposal, which allows the developers to control NAO on a more basic level than the included Choreographe program. The SDK is compatible with several programming languages as shown in Figure 2.1.

2.2.1 Python

Python is an interpreted high-level programming language for general-purpose programming. Created by Guido van Rossum and first released in 1991, Python has a design philosophy that emphasizes code readability, notably using significant whitespace. It provides constructs that enable clear programming on both small and large scales.[24]

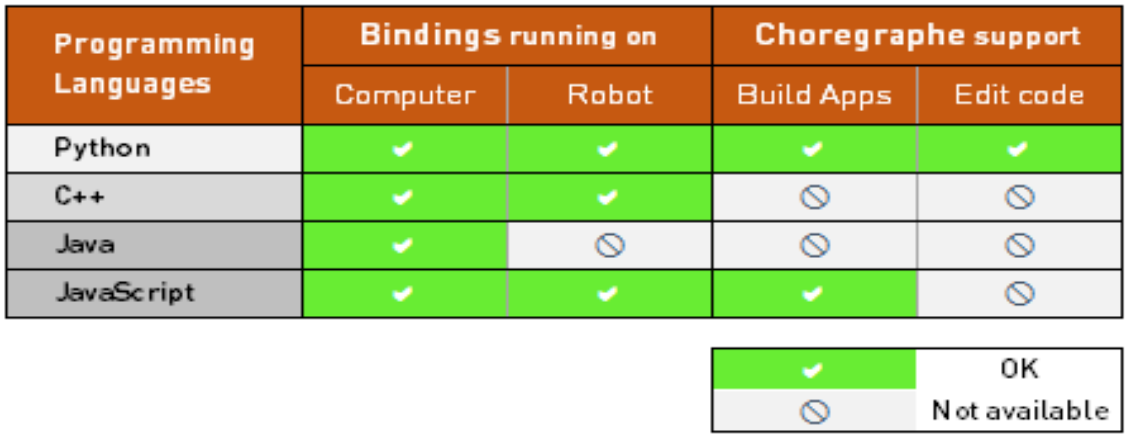


Figure 2.1: Programming languages. (Image from Aldebaran's documentation)

Python features a dynamic type system and automatic memory management. It supports multiple programming paradigms, including object-oriented, imperative, functional and procedural, and has a large and comprehensive standard library.

A simple example of a remote module using a memory proxy is given in Listings 2.1. This short program firstly connects to Naoqi running on the robot on port 9559 and IP address specified in robot-IP. Then it inserts a value 3.14 associated with name "myValueName" into the shared memory using the memory proxy.

Listing 2.1: Example using Python on Nao.

```
from naoqi import ALProxy
memProxy = ALProxy('ALMemory,robot-IP,9559)
memProxy.insertData('myValueName, 3.14)
```

2.3 Website Development

To inform the rescuers about the current situation which our humanoid robot is performing, we created a website using PHP, HTML, CSS, and JavaScript.

2.3.1 HTML

Hypertext Markup Language (HTML) is the standard markup language for creating web pages and web applications.

Web browsers receive HTML documents from a web server or from local storage and render the documents into multimedia web pages. HTML describes the structure

of a web page semantically and originally included cues for the appearance of the document.

HTML elements are the building blocks of HTML pages. With HTML constructs, images and other objects such as interactive forms may be embedded into the rendered page. HTML provides a means to create structured documents by denoting structural semantics for the text such as headings, paragraphs, lists, links, quotes and other items. HTML elements are delineated by tags, written using angle brackets. Tags such as <img> and <input> directly introduce content into the page. Other tags such as surround and provide information about document text and may include other tags as sub-elements. Browsers do not display the HTML tags but use them to interpret the content of the page.

Listing 2.2: HTML example.

```
<!DOCTYPE html>
<html>
<head>
<title>Page Title</title>
</head>
<body>

<h1>This is a Heading</h1>
<p>This is a paragraph.</p>

</body>
</html>
```

2.3.2 CSS

Cascading Style Sheets (CSS) is a style sheet language used for describing the presentation of a document written in a markup language like HTML.

CSS is designed to enable the separation of presentation and content, including layout, colors, and fonts [25]. This separation can improve content accessibility, provide more flexibility and control in the specification of presentation characteristics,

enable multiple web pages to share formatting by specifying the relevant CSS in a separate .css file and reduce complexity and repetition in the structural content.

Separation of formatting and content also makes it feasible to present the same markup page in different styles for different rendering methods, such as on-screen, in print, by voice (via speech-based browser or screen reader), and on Braille-based tactile devices. CSS also has rules for alternate formatting if the content is accessed on a mobile device.

The name cascading comes from the specified priority scheme to determine which style rule applies if more than one rule matches a particular element.

Listing 2.3: CSS example.

```
body {
    background-color: lightblue;
}

h1 {
    color: white;
    text-align: center;
}

p {
    font-family: verdana;
    font-size: 20px;
}
```

2.3.3 JavaScript

JavaScript often abbreviated as JS, is a high-level, interpreted programming language [26]. It is a language which is also characterized as dynamic, weakly typed, prototype-based and multi-paradigm.

Alongside HTML and CSS, JavaScript is one of the three core technologies of the World Wide Web [26]. JavaScript enables interactive web pages and thus is an

essential part of web applications. The vast majority of websites use it and all major web browsers have a dedicated JavaScript engine to execute it.

As a multi-paradigm language, JavaScript supports event-driven, functional, and imperative (including object-oriented and prototype-based) programming styles. It has an API for working with text, arrays, dates, regular expressions, and basic manipulation of the DOM, but the language itself does not include any I/O, such as networking, storage, or graphics facilities, relying for these upon the host environment in which it is embedded.

Listing 2.4: JavaScript example.

```
<script>
var FIRSTvariable = window.prompt("PLEASE FILL IN YOUR NAME")
alert("Your name is " + FIRSTvariable + ".")
</script>
```

2.3.4 PHP

PHP, Hypertext Preprocessor (or simply PHP), is a server-side scripting language designed for Web development, but also used as a general-purpose programming language. It was originally created by Rasmus Lerdorf in 1994, [27]. PHP originally stood for Personal Home Page, [27] but it now stands for the recursive initialism PHP: Hypertext Preprocessor.

PHP code may be embedded into HTML code, or it can be used in combination with various web template systems, web content management systems, and web frameworks. The PHP code is usually processed by a PHP interpreter implemented as a module in the web server or as a Common Gateway Interface (CGI) executable. The web server combines the results of the interpreted and executed PHP code, which may be any type of data, including images, with the generated web page. PHP code may also be executed with a command-line interface (CLI) and can be used to implement standalone graphical applications [28].

Listing 2.5: PHP example.

```
<!DOCTYPE html>
<html>
```

```
<body>

<?php
echo "My first PHP script!";
?>

</body>
</html>
```

2.4 Socket Programming

We used TCP socket programming for the reason that it is necessary for our system to use a protocol with small communication overhead for avoiding high latency responses.

2.4.1 Description

Sockets programming is the fundamental technology behind communications on TCP/IP networks. A socket is one endpoint of a two-way link between two programs running on a network. The socket provides a bidirectional communication endpoint for sending and receiving data with another socket. Socket connections normally run between two different computers on a local area network (LAN) or across the internet, but it can also use them for interprocess communication on a single computer.

2.4.2 Sockets and Addresses

Socket endpoints on TCP/IP networks each have a unique address that is the combination of an IP address and a TCP/IP port number. Because the socket is bound to a specific port number, the TCP layer can identify the application that should receive the data sent to it. When creating a new socket, the socket library automatically generates a unique port number on that device. The programmer can also specify port numbers in specific situations.

2.4.3 How Server Sockets Work

The server has a socket that is bound to a specific port. The server waits for a different computer to make a connection request. The client computer knows the hostname of the server computer and the port number on which the server is listening. The client computer identifies itself, and—if everything goes right—the server permits the client computer to connect.

2.4.4 TCP Socket Flow

The diagram which is depicted in Figure 2.2 represents the sequence of socket API calls and data flow for TCP.

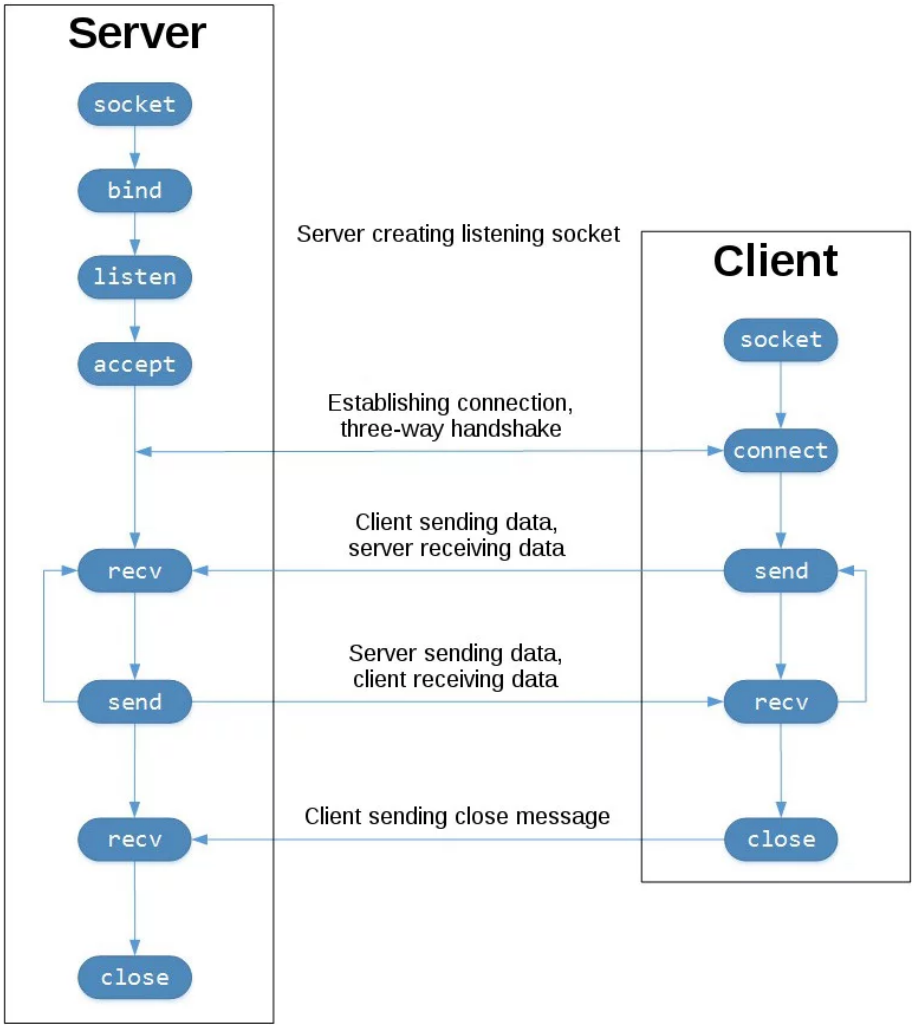


Figure 2.2: TCP Socket Flow.

The left-hand column represents the server. On the right-hand side is the client. Starting in the top left-hand column, note the API calls the server makes to set up a

"listening" socket:

- socket()
- socket()
- bind()
- listen()
- accept()

A listening socket does just what it sounds like. It listens for connections from clients. When a client connects, the server calls accept() to accept, or complete, the connection.

The client calls connect() to establish a connection to the server and initiate the three-way handshake. The handshake step is important since it ensures that each side of the connection is reachable in the network, in other words, that the client can reach the server and vice versa. It may be that only one host, client or server can reach the other.

In the middle is the round-trip section, where data is exchanged between the client and server using calls to send() and recv().

At the bottom, the client and server close() their respective sockets.

2.4.5 TCP Communication on Python

- **First step** is to import the socket module

Listing 2.6: Importing module socket.

```
import socket
```

- **Second step** is to create the socket and then the connection.

Listing 2.7: Client program example.

```
# create an INET, STREAMing socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((HOST, PORT))
```

Listing 2.8: Server program example.

```
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((HOST, PORT))
s.listen(1)
conn, addr = s.accept()
```

2.5 Advanced Encryption Standard (AES)

In order to have a secure system, we made an encrypted communication between our humanoid robot and the server. Specifically, we used AES-128.

2.5.1 Cryptography

Cryptography is a practice and study of techniques for secure communication. The basic elements of a cryptographic algorithm are plaintext, key, and ciphertext. The data which is present in its natural format is known as plaintext. The key is a sequence that controls the behavior of the algorithm. Ciphertext is the data which is unreadable by anyone expect the intended recipients. The modern field of cryptography includes symmetric key and asymmetric key cryptographic algorithms [29].

Symmetric key algorithms use the same key for both encryptions of plaintext and decryption of ciphertext. Across the years, various comprehensive data encryption techniques have been developed. Some popular examples of symmetric key algorithms include RC4 (Rivest Cipher 4), DES (Data Encryption Standard), AES and triple DES [29]. AES cipher also known as Rijndael cipher is the most advanced cryptographic algorithm. [30].

In the United States, AES was announced by the NIST as U.S. FIPS PUB 197 (FIPS 197) on November 26, 2001 [30]. This announcement followed a five-year standardization process in which fifteen competing designs were presented and evaluated before the Rijndael cipher was selected as the most suitable.

2.5.2 AES Algorithm

AES is a symmetric key block cipher existing in various key lengths of 128-bit, 192-bit, and 256-bit. A block cipher usually consists of two paired algorithms, one for

encryption at sender side and other for decryption at the receiver side. As AES is a symmetric key cipher both sender and receiver share the same key which is also known as the private key. With the message P and the key K as input, the encryption algorithm forms the ciphertext C as mentioned below.

$$C = E_K(P)$$

The notation as represented above indicates that the ciphertext C is produced by using encryption algorithm E, as a function of the plaintext P, with the specific function determined by the value of the key K [31]. The intended receiver in possession of the key is able to invert the transformation and retrieve the plain text.

The inverse transformation that is performed using decryption algorithm D as a function of the ciphertext C [31] is represented below.

$$P = D_K(C)$$

AES is an iterative cipher comprising computational rounds for both encryption and decryption. For every additional 32 bits in cipher key, the number of rounds is increased by one [32]. The number of rounds for various lengths of AES ciphers is given in Table 2.1. AES 128-bit cipher has been used in the proposed design.

Table 2.1: AES Structure.

Classification	**Rounds**
AES-128	10
AES-192	12
AES-256	14

The first round process is depicted in Figure 2.3.

2.5.3 Security

Until May 2009, the only successful published attacks against the full AES were side-channel attacks on some specific implementations. The National Security Agency (NSA) reviewed all the AES finalists, including Rijndael, and stated that all of them were secure enough for U.S. Government non-classified data. In June 2003, the U.S. Government announced that AES could be used to protect classified information [33].

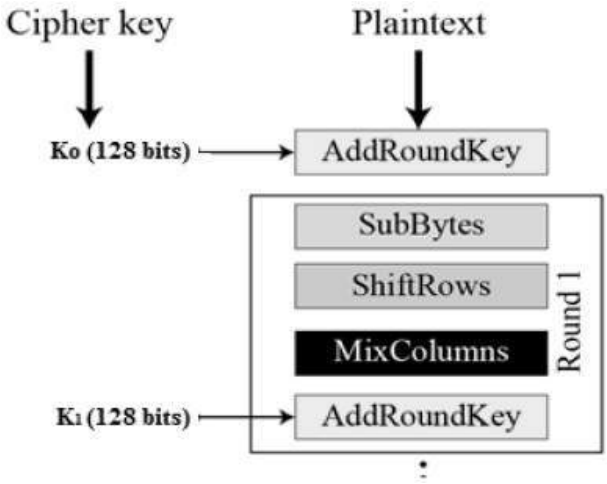


Figure 2.3: Encryption Process.

The design and strength of all key lengths of the AES algorithm are sufficient to protect classified information up to the SECRET level. TOP SECRET information will require the use of either the 192 or 256 key lengths. The implementation of AES in products intended to protect national security systems and/or information must be reviewed and certified by NSA prior to their acquisition and use [33].

2.5.4 AES and Python

PyCrypto is a library, which provides secure hash functions and various encryption algorithms. It supports Python version 2.1 through 3.3.

To install the library PyCrypto you use the command in Listing 2.9.

Listing 2.9: Installation of PyCrypto.

```
$ pip install pycrypto
```

Listing 2.10: AES Example.

```
from Crypto.Cipher import AES
# Encryption
encryption_suite = AES.new('This is a key123', AES.MODE_CBC, '
   This is an IV456')
cipher_text = encryption_suite.encrypt("A really secret message
   . Not for prying eyes.")

# Decryption
```

```
decryption_suite = AES.new('This is a key123', AES.MODE_CBC, '
   This is an IV456')
plain_text = decryption_suite.decrypt(cipher_text)
```

2.6 OpenCV

OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. OpenCV is released under a BSD license (a family of permissive free software licenses, imposing minimal restrictions on the redistribution of covered software) and hence it's free for both academic and commercial use. It has C++, C, Python and Java interfaces and supports Windows, Linux, Mac OS, iOS and Android. OpenCV was designed for computational efficiency and with a strong focus on real-time applications [34].

2.6.1 Basics

Object Detection using Haar feature-based cascade classifiers is an effective object detection method proposed by Paul Viola and Michael Jones [35]. It is a machine learning based approach where a cascade function is trained from a lot of positive and negative images. It is then used to detect objects in other images.

In this thesis, we worked with face detection. Initially, the algorithm needs a lot of positive images (images of faces) and negative images (images without faces) to train the classifier. Then we need to extract features from it. For this, Haar features shown in the Figure 2.4 are used. They are just like our convolutional kernel. Each feature is a single value obtained by subtracting the sum of pixels under the white rectangle from the sum of pixels under the black rectangle.

2.6.2 Haar-cascade Detection in OpenCV

OpenCV comes with a trainer as well as a detector.

In this thesis, we will deal with detection. OpenCV already contains many pre-trained classifiers for face, eyes, smiles, etc. Those XML files are stored in the

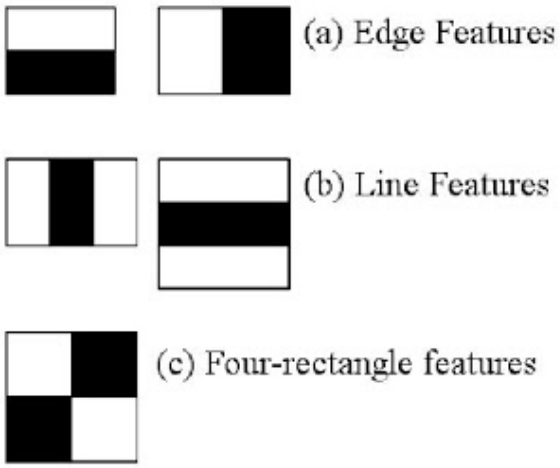


Figure 2.4: Haar features.

OpenCV/data/haarcascades/ folder.

To create a face and eye detector with OpenCV first, we need to load the required XML classifiers. Then load our input image (or video) in grayscale mode.

Listing 2.11: Loading XML classifiers.

```
import numpy as np
import cv2 as cv
face_cascade = cv.CascadeClassifier('
   haarcascade_frontalface_default.xml')
eye_cascade = cv.CascadeClassifier('haarcascade_eye.xml')
img = cv.imread('sachin.jpg')
gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
```

Now we find the faces in the image. If faces are found, it returns the positions of detected faces as Rect(x,y,w,h). Once we get these locations, we can create a region of interest (ROI) for the face and apply eye detection on this ROI.

Listing 2.12: Detecting Faces.

```
faces = face_cascade.detectMultiScale(gray, 1.3, 5)
for (x,y,w,h) in faces:
    cv.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)
    roi_gray = gray[y:y+h, x:x+w]
    roi_color = img[y:y+h, x:x+w]
    eyes = eye_cascade.detectMultiScale(roi_gray)
```

```
    for (ex,ey,ew,eh) in eyes:
        cv.rectangle(roi_color,(ex,ey),(ex+ew,ey+eh),(0,255,0),
            2)
cv.imshow('img',img)
cv.waitKey(0)
cv.destroyAllWindows()
```

The result looks like the image depicted in Figrue 2.5.

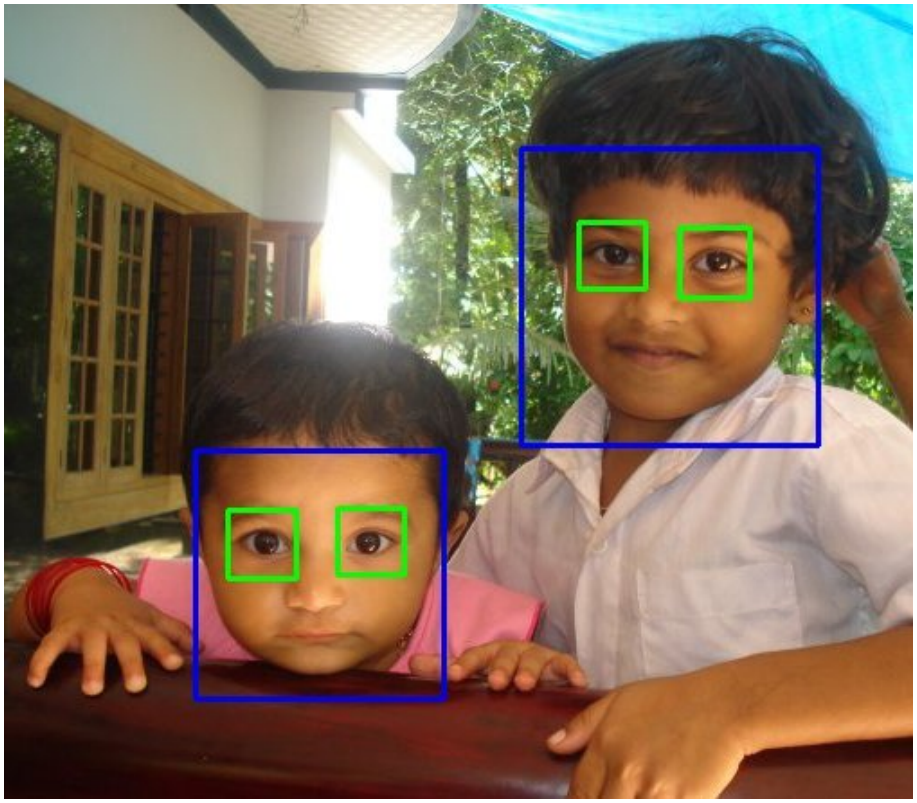


Figure 2.5: Face Detection on OpenCV.

2.7 Chapter Summary

In chapter 2 we discussed the tools that help us in order to create the code of our system but also the tools we needed in order to connect to our cloud server. Afterward, we continued by introducing the choice of programming languages and software for this thesis. Some of the choices regarding the socket programming and the encryption for the communication was explained. Chapter 2 ends with some necessary theory related to OpenCV and image processing.

Chapter 3

Implementation Details

3.1 System Architecture

The overall system architecture of my thesis is depicted in Figure 3.1. Our system comprises a humanoid robot, a cloud server and a secure website for the rescuer. Our humanoid robot communicates with the server so internet connection is mandatory in order to autonomously move even in non-routable networks consisting of RFC 1918 Private Internets.

3.2 Proxies on Nao

A proxy is an object that will behave as the module it represents.

For instance, if you create a proxy to the ALMotion module, you will get an object containing all the ALMotion methods.

In our case we created 4 proxies, see Listing 3.1. Specifically:

- **A proxy to the ALMotion module.**
  ALMotion provides methods that help make the robot move. It contains four major groups of methods for controlling the:
  - joint stiffness (basically motor On-Off),
  - joint position (interpolation, reactive control),
  - walk (distance and velocity control, world position and so on),
  - robot effector in the Cartesian space (inverse kinematics, whole body constraints).

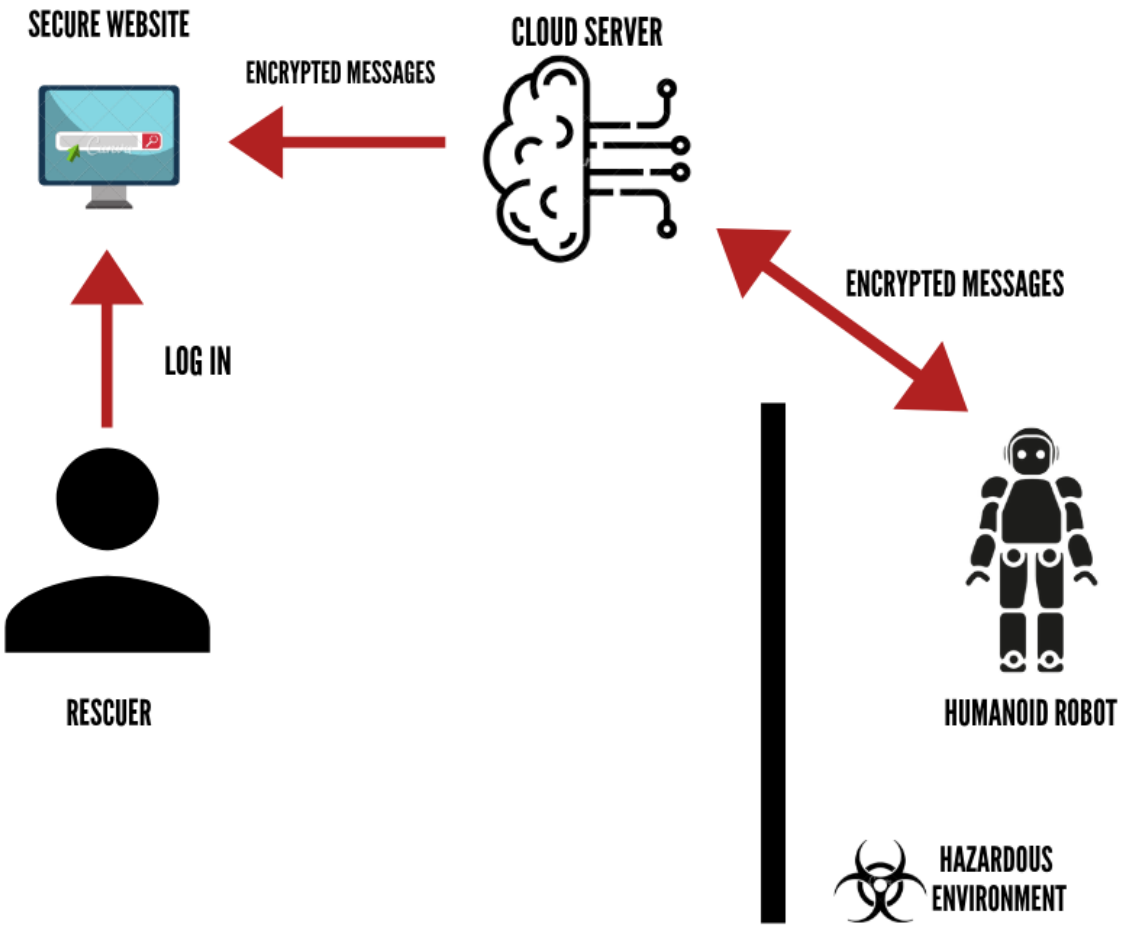


Figure 3.1: System architecture.

- **A proxy to the ALRobotPosture module.**
  ALRobotPosture module allows you to make the robot go to different predefined postures.

- **A proxy to the ALMemory module.**
  ALMemory is a centralized memory used to store all key information related to the hardware configuration of your robot.

- **A proxy to the ALSonar module.**
  ALsonar module retrieves ultrasonic sensor value from ALMemory, process it and raises events according to the situation.

- **A proxy to the ALAudioRecorder module.**
  ALAudioRecorder provides recording services in “WAV” and “OGG” file format of the signals coming from NAO’s microphones.

- **A proxy to the ALPhotoCapture module.**
  The ALPhotoCapture module allows you to take pictures and using the robot cameras and save them on disk.

Listing 3.1: Proxies.

```
try:
        motion = ALProxy("ALMotion", "127.0.0.1", 9559)
except Exception, a:
        print "Could not create proxy to ALMotion"
        print "Error was: ", a
#============================================================
try:
        postureProxy = ALProxy("ALRobotPosture", "127.0.0.1", 9
           559)
except Exception, b:
        print "Could not create proxy to ALRobotPosture"
        print "Error was: ", b
#============================================================
try:
        sonarProxy = ALProxy("ALSonar", "127.0.0.1", 9559)
except Exception, c:
        print "Could not create proxy to ALSonar"
        print "Error was: ", c
#============================================================
try:
        memoryProxy = ALProxy("ALMemory", "127.0.0.1", 9559)
except Exception, d:
        print "Could not create proxy to ALMemory"
        print "Error was: ", d
```

3.3 Motion on Nao

There are 3 high-level ways to control the locomotion (Figure 3.2).

Table 3.1: Locomotion.

Use	To set
ALMotionProxy::moveTo()	a target pose on the ground plane,.
ALMotionProxy::move()	the robot's instantaneous velocity in SI units.
ALMotionProxy::moveToward()	the robot's instantaneous normalized velocity .

For even more control over the walk, we may also use ALMotionProxy::setFootSteps() (Listing 3.2)or ALMotionProxy::setFootStepsWithSpeed(), to provide the exact footsteps NAO will follow.

Listing 3.2: Method setFootSteps.

```
void ALMotionProxy::setFootSteps(const std::vector<std::string
   >& legName, const AL::ALValue& footSteps, const std::vector<
   float>& timeList, const bool& clearExisting)
```

Makes the robot do foot step planner. This is a non-blocking call.
Parameters:

- legName – name of the leg to move('LLeg'or 'RLeg').
- footSteps – [x, y, theta], [Position along X/Y, Orientation round Z axis] of the leg relative to the other Leg in [meters, meters, radians]. Must be less than [MaxStepX, MaxStepY, MaxStepTheta]
- timeList – time list of each foot step.
- clearExisting – Clear existing foot steps.

3.3.1 Performance and Limitations

NAO's walk is stabilized using feedback from his joint sensors. This makes the walk robust against small disturbances and absorbs torso oscillations in the frontal and lateral planes.

NAO is able to walk on multiple floor surfaces such as carpet, tiles and wooden floors.

He can transition between these surfaces while walking. However, large obstacles can still make him fall, as he assumes that the ground is more or less flat.

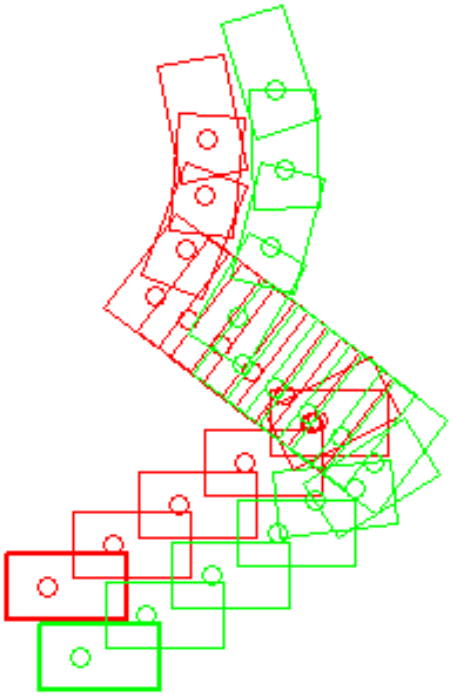


Figure 3.2: Locomotion Control.

3.4 Obstacle Detection

It is necessary for our humanoid robot to detect obstacles in order to move properly in space. For this reason, we created two functions, one using **ultrasonic sensors** and also another one using **Nao's hands**, for detecting obstacles.

3.4.1 Obstacle Detection using ultrasonic sensors

NAO is equipped with two ultrasonic sensors (or sonars), Figure 3.3, which allow it to estimate the distance to obstacles in its environment. The detection range goes from 25 cm to 255 cm, but under 25 cm there is no distance information, the robot only knows that an object is present.

Specifications:

- Frequency: 40kHz
- Resolution: 1cm-4cm (depending on distance)

- Detection range: 0.20 m - 0.80 m
  - Under 20 cm there is no distance information, the robot only knows that an object is present.
  - Above 80 cm the value returned is an estimation. For further details, read the important tips in US/Sensor (m).
- Effective cone: 60°

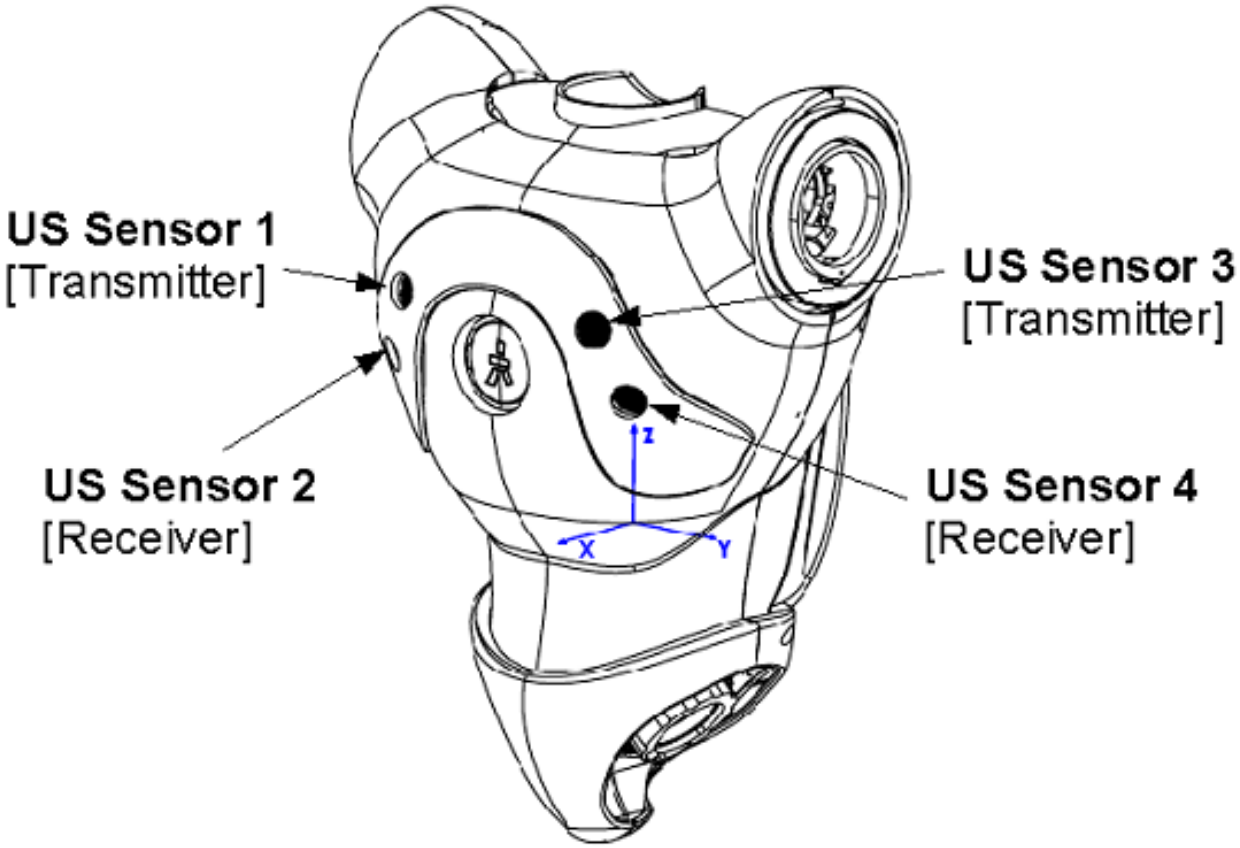


Figure 3.3: Sonars.

In the Listing 3.3 you can see the code we use for obstacle detection using sonar in python.

Listing 3.3: Obstacle-sonar Function.

```
def object_front(robotIp):

#Function for checking objects in front
```

```
    # Subscribe to sonars, this will launch sonars (at hardware
        level) and start data acquisition.
    sonarProxy.subscribe("myApplication")
    # Get sonar left first echo (distance in meters to the
       first obstacle).
    Left = memoryProxy.getData("Device/SubDeviceList/US/Left/
       Sensor/Value")
    # Same thing for right.
    Right = memoryProxy.getData("Device/SubDeviceList/US/Right/
       Sensor/Value")
    #print Sonar Left and Right
    print( "Left : %.2f" %  (Left) )
    print( "Right : %.2f" %  (Right) )
    return Left, Right
```

3.4.2 Obstacle Detection using Hands

Apart from detecting obstacles in front, it is crucial to detect obstacles on the side of the robot. So, we needed to think another way for the detection because Function 3.3 was really inefficient.

For this reason, we used the hands of our humanoid robot. Specifically, as you can see in Figure 3.4, Nao lifts up its hands to check for obstacles. In the Listing 3.4 you can see the code we use for obstacle detection using hansd in python.

Listing 3.4: Obstacle on the side Function.

```
def object_hand():

#Function for checking objects left and right using hands

    motion.setStiffnesses("RArm", 1.0)
    motion.setStiffnesses("LArm", 1.0)
```

```
# Example showing a single target angle for one joint
# Interpolate the head yaw to 1.0 radian in 1.0 second
names      = ["LShoulderRoll","RShoulderRoll"]
angleLists = [76.0*almath.TO_RAD, -76.0*almath.TO_RAD]
timeLists  = 1.0
isAbsolute = True
motion.angleInterpolation(names, angleLists, timeLists,
   isAbsolute)
time.sleep(0.5)
useSensors  = True
sensorAngles = motion.getAngles(names, useSensors)
ar=float(str(sensorAngles[0]))
de=float(str(sensorAngles[1]))
print( "Left_Hand : %.2f" %  (ar) )
print( "Right_Hand : %.2f" %  (de) )
return ar, de
```

3.5 Movement Functions on Nao

It is crucial for our humanoid robot to be able to autonomously move in the space. So, we created four functions which it calls them where appropriate.

- **Moving Forward**. The code in python is located in Listing 3.5.
- **Moving Left.** If the obstacle is located as shown in figure 3.3, NAO uses the moving left function. The code in python is located in Listing 3.6.
- **Moving Right.** If the obstacle is located as shown in figure 3.4, NAO uses the moving right function. The code in python is located in Listing 3.7.
- **Turning Left.** If the obstacle is located as shown in figure 3.5, NAO uses the turning left function. The code in python is located in Listing 3.8.
- **Turning Right.** If the obstacle is located as shown in figure 3.6, NAO uses the turning right function. The code in python is located in Listing 3.9.

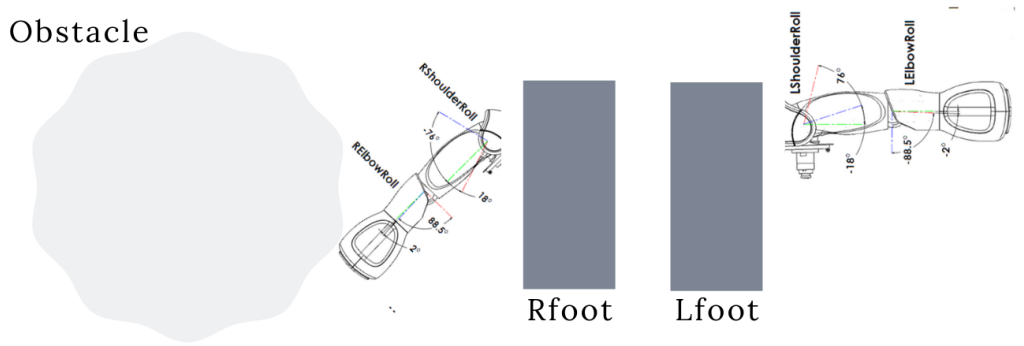


Figure 3.4: Detecting obstacles using hands.

- **Turning Back.** If the obstacle is located as shown in figure 3.7, NAO uses the turning back function. The code in python is located in Listing 3.10.

Listing 3.5: Moving Forward function.

```
def  move_forward():
#Function for moving forward

        # A small step forwards and anti-clockwise with the
           left foot
        legName  = ["LLeg"]
        X        = 0.5
        Y        = 0.0
        Theta    = 0.0
        footSteps = [[X, Y, Theta]]
        timeList = [0.3]
        clearExisting = False
```

```
        motion.setFootSteps(legName, footSteps, timeList,
            clearExisting)
        time.sleep(1.0)

        # A small step forwards and anti-clockwise with the
            right foot
        legName  = ["RLeg"]
        X        = 0.5
        Y        = 0.0
        Theta    = 0.0
        footSteps = [[X, Y, Theta]]
        timeList = [0.3]
        clearExisting = False
        motion.setFootSteps(legName, footSteps, timeList,
            clearExisting)
        time.sleep(2.0)
        Left = memoryProxy.getData("Device/SubDeviceList/US/
            Left/Sensor/Value")
        Right = memoryProxy.getData("Device/SubDeviceList/US/
            Right/Sensor/Value")
```

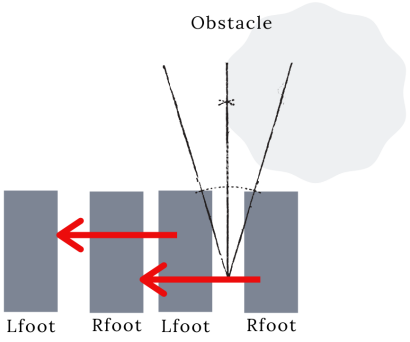


Figure 3.5: Moving Left.

Listing 3.6: Moving Left function.

```
def for_left():

#Function for moving left

        legName  = ["LLeg"]
                X        = 0.0
                Y        = 0.5
                Theta    = 0.0
                footSteps = [[X, Y, Theta]]
                timeList = [0.6]
                clearExisting = False
                motion.setFootSteps(legName, footSteps,
                   timeList, clearExisting)
                time.sleep(1.0)

                # Right Foot Now
                legName  = ["RLeg"]
                X        = 0.0
                Y        = 0.5
                Theta    = 0.0
                footSteps = [[X, Y, Theta]]
                timeList = [0.6]
                clearExisting = False
                motion.setFootSteps(legName, footSteps,
                   timeList, clearExisting)
```

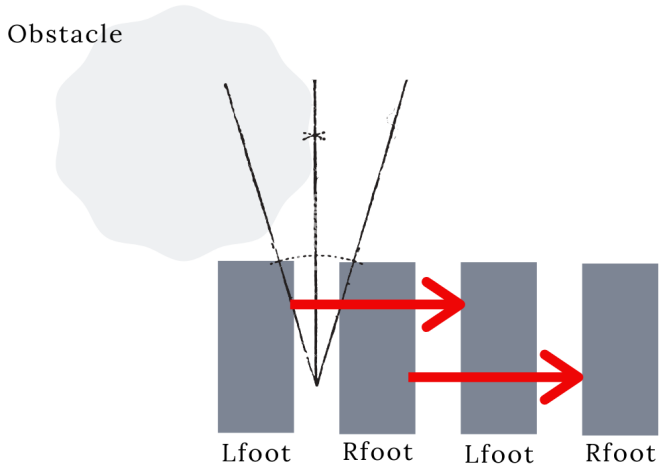


Figure 3.6: Moving Right.

Listing 3.7: Moving Right function.

```
def for_right():

#Function for moving right

            legName   = ["RLeg"]
                X         = 0.0
                Y         = -0.5
                Theta     = 0.0
                footSteps = [[X, Y, Theta]]
                timeList = [0.6]
                clearExisting = False
                motion.setFootSteps(legName, footSteps,
                   timeList, clearExisting)
                time.sleep(1.0)

                # Left Foot Now
```

```
                legName   = ["LLeg"]
                X         = 0.0
                Y         = -0.5
                Theta     = 0.0
                footSteps = [[X, Y, Theta]]
                timeList = [0.6]
                clearExisting = False
                motion.setFootSteps(legName, footSteps,
                   timeList, clearExisting)
```

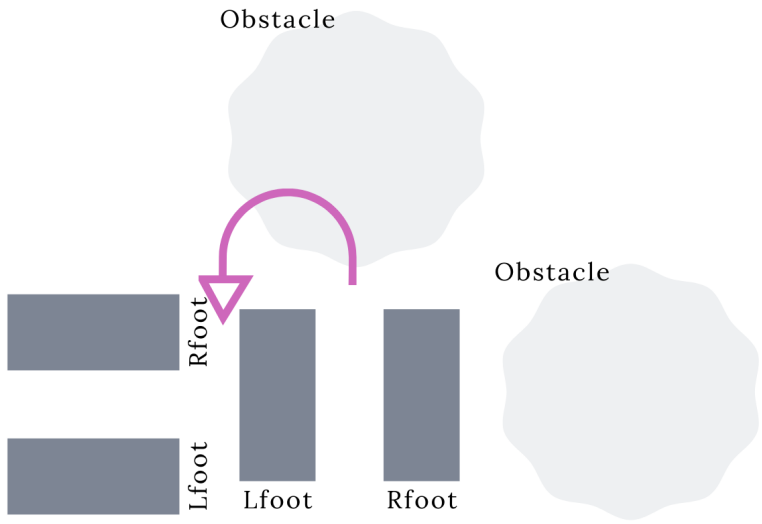


Figure 3.7: Turning Left.

Listing 3.8: Turning Left function.

```
def left():

#Function turning left

        X = 0
        Y = 0
```

```
        Theta = math.pi/2.0
        Frequency=0
        motion.post.moveTo(X, Y, Theta)
        # wait that the move process start running
        time.sleep(0.1)

        footSteps1 = motion.getFootSteps()
        # Second call of move API
        motion.post.moveTo(X, Y, Theta)
        # get the second foot steps vector
        footSteps2 = motion.getFootSteps()
        # here we wait until the move process is over
        motion.waitUntilMoveIsFinished()
        # then we get the final robot position
```

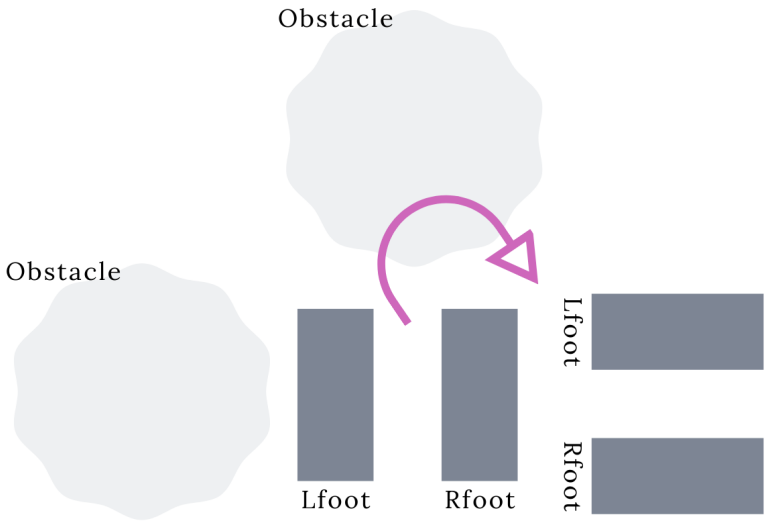


Figure 3.8: Turning Right.

Listing 3.9: Turning Right function.

```
def right():
```

```
#Function turning right

        X = 0
        Y = 0
        Theta = -math.pi/2.0
        Frequency=0
        motion.post.moveTo(X, Y, Theta)
        # wait that the move process start running
        time.sleep(0.1)

        footSteps1 = motion.getFootSteps()
        # Second call of move API
        motion.post.moveTo(X, Y, Theta)
        # get the second foot steps vector
        footSteps2 = motion.getFootSteps()
        # here we wait until the move process is over
        motion.waitUntilMoveIsFinished()
        # then we get the final robot position
```

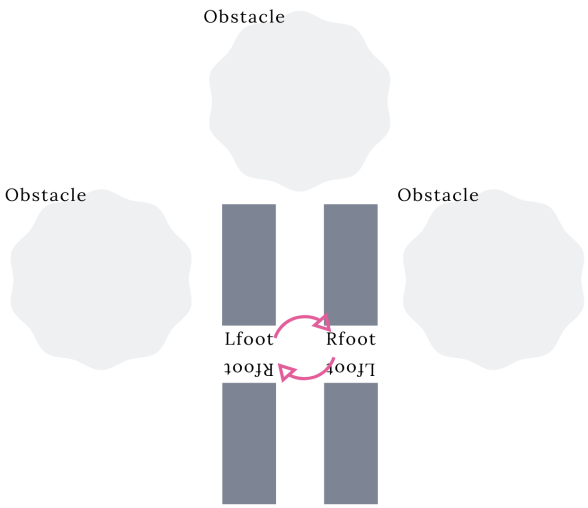


Figure 3.9: Turning Back.

Listing 3.10: Turning Back function.

```
def back():

#Function moving 180 degrees

        X = 0
        Y = 0
        Theta = -math.pi
        Frequency=0
        motion.post.moveTo(X, Y, Theta)
        # wait that the move process start running
        time.sleep(0.1)
        footSteps1 = motion.getFootSteps()
        # Second call of move API
        motion.post.moveTo(X, Y, Theta)
        # get the second foot steps vector
        footSteps2 = motion.getFootSteps()
        # here we wait until the move process is over
        motion.waitUntilMoveIsFinished()
        # then we get the final robot position
```

3.6 Photo Capturing from Nao

The ALPhotoCapture module allows you to take pictures and using the robot cameras and save them on disk.

Listing 3.11: Photo Capturing function.

```
def camera():

#Function for capturing a photo
```

```
        # Create a proxy to ALPhotoCapture
        try:
          photoCaptureProxy = ALProxy("ALPhotoCapture", "127.0.
             0.1", 9559)
        except Exception, e:
          print "Error when creating ALPhotoCapture proxy:"
          print str(e)
          exit(1)
        photoCaptureProxy.setResolution(2)
        photoCaptureProxy.setPictureFormat("jpg")
        photoCaptureProxy.takePictures(1, "/var/volatile/", "
           image")
```

3.7 Audio Recording from Nao

ALAudioRecorder relies on the Linux library SNDFile to efficiently encode audio inputs in real time. ALAudioRecorder collects input signals through ALAudioDevice.

Listing 3.12: Turning Back function.

```
def audio():

#Function for audio player and for recording
        tts = ALProxy("ALTextToSpeech", "127.0.0.1", 9559)
        audio = ALProxy("ALAudioDevice", "127.0.0.1", 9559)
        record = ALProxy("ALAudioRecorder", "127.0.0.1", 9559)
        aup = ALProxy("ALAudioPlayer", "127.0.0.1", 9559)



        tts.setParameter("doubleVoice", 1)
        tts.setParameter("doubleVoiceLevel", 0)
        tts.setParameter("doubleVoiceTimeShift", 0.1)
```

```
        tts.setParameter("pitchShift", 1.1)
        tts.say("Do you need Help?")
        record.startMicrophonesRecording("/var/volatile/test.
           wav", 'wav', 16000, (0,0,1,0))
        time.sleep(3)
        record.stopMicrophonesRecording()
        tts.say("Recording is. over.")
        time.sleep(1)
```

3.7.1 Performances and Limitations

The recording capabilities are limited to the following formats:

- four channels 48000Hz in OGG.
- four channels 48000Hz in WAV.
- one channels (front, rear, left or right), 16000Hz, in OGG.
- one channels (front, rear, left or right), 16000Hz, in WAV.

3.8 Cloud Server

The cloud server is the brain of our system. As we mentioned previously, the humanoid robot communicates encrypted with our server sending precious information. When the user-rescuer enables the robot, the cloud server sends a signal to begin.

In the beginning, our robot is checking the environment for obstacles and sends back to the server important values. With these values, the server decides whether the robot moves forward or it is necessary to use the camera. If there is an obstacle in front, the server sends an encrypted message to our robot demanding it to use the camera and capture a photo. The robot sends encrypted using socket programming (as we mention in the previous section) the photo following the last demand from the server. As long as the server receives the photo uses OpenCV for processing the image. After that, if our server does not detect a person on the image sends to Nao the necessary move in order to avoid the obstacle.

This is a repeated procedure unless the server detects a person on the image or the rescuer disables the robot as you can see in Figure 3.10. For more details about the code you can see Appendix A.

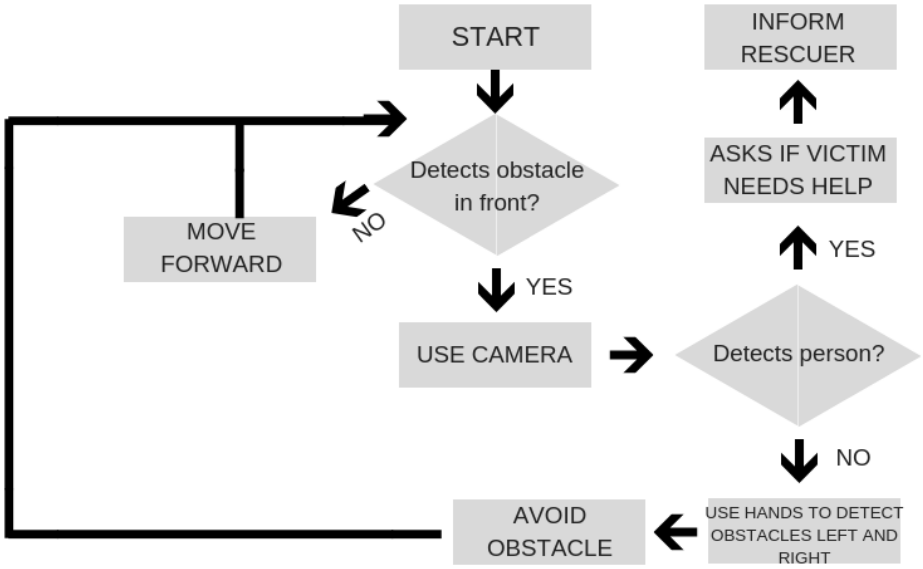


Figure 3.10: Block Diagram.

3.8.1 Speech Recognition

Google has a great Speech Recognition API. This API converts spoken text (microphone) into written text (Python strings), briefly Speech to Text. You can simply speak in a microphone and Google API will translate this into written text. The API has excellent results for the English language.

Listing 3.13: Processing Sound using Google Speech Recognition.

```
r = sr.Recognizer()
with sr.WavFile("voice.wav") as source:
# use "test.wav" as the audio source
                audio = r.record(source)
                # extract audio data from the file


try:
                print( r.recognize_google(audio))
                # recognize speech using Google Speech
                   Recognition
        if "yes" in  r.recognize_google(audio):
                t = open("/home/user/public_html/panel/person.
                   txt", "w+")
                        t.write('1')
                        t.close()


        else:
                t = open("/home/user/public_html/panel/person.
                   txt", "w+")
                        t.write('2')
                        t.close()



except LookupError:
# speech is unintelligible
        print("Could not understand audio")
break
```

The program in the Listing 3.13 takes a recorded audio, send it to the speech API and return a Python string.

The audio is recorded using the speech recognition module, the module will

include on top of the program. Secondly, we send the recorded speech to the Google speech recognition API which will then return the output. r.recognize_google(audio) returns a string.

3.9 Website

As we mention above the security is fundamental for our system. So, in order the user-rescuer to view information from the website the first step is to log in with the given credentials.

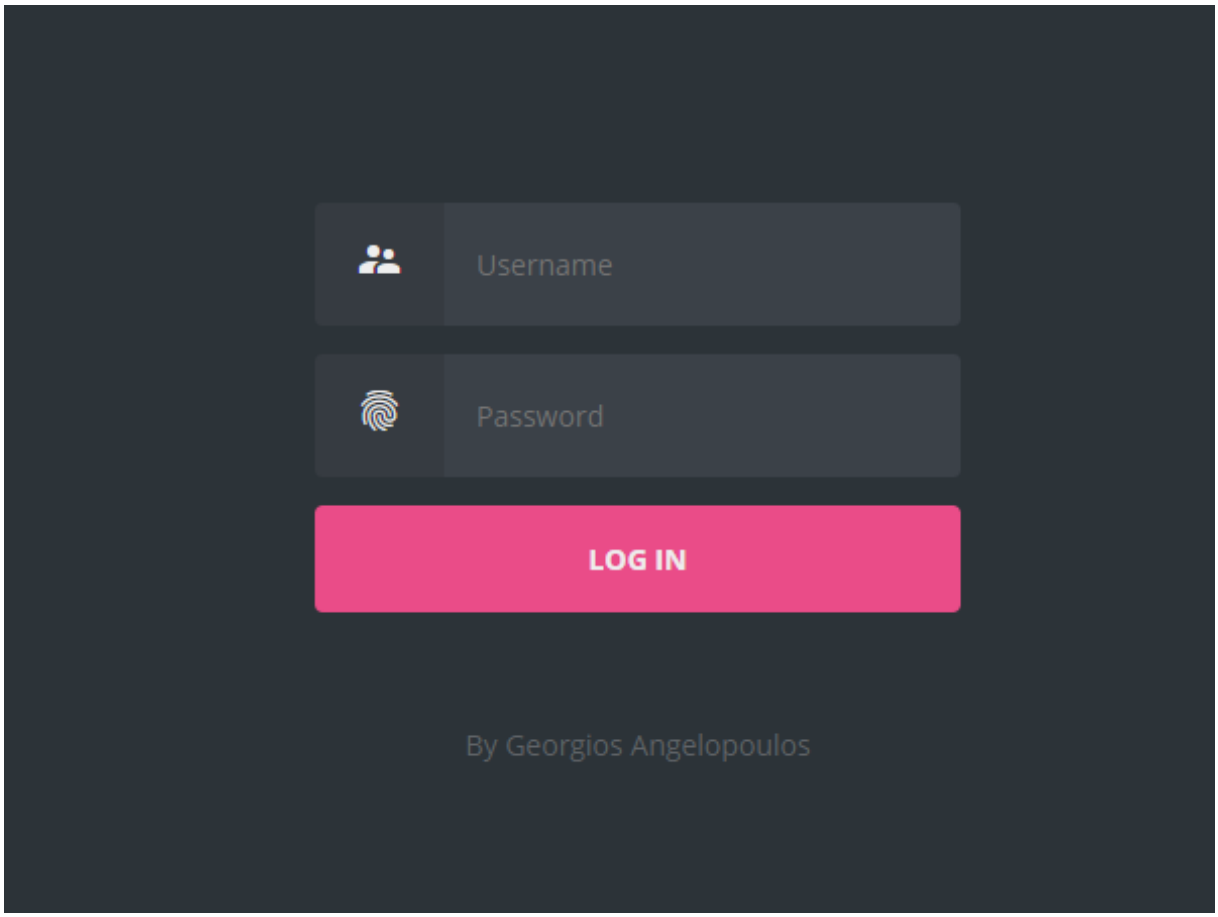


Figure 3.11: Log in Page.

As long as the user inserts the correct credentials in the login page, can view the control panel.

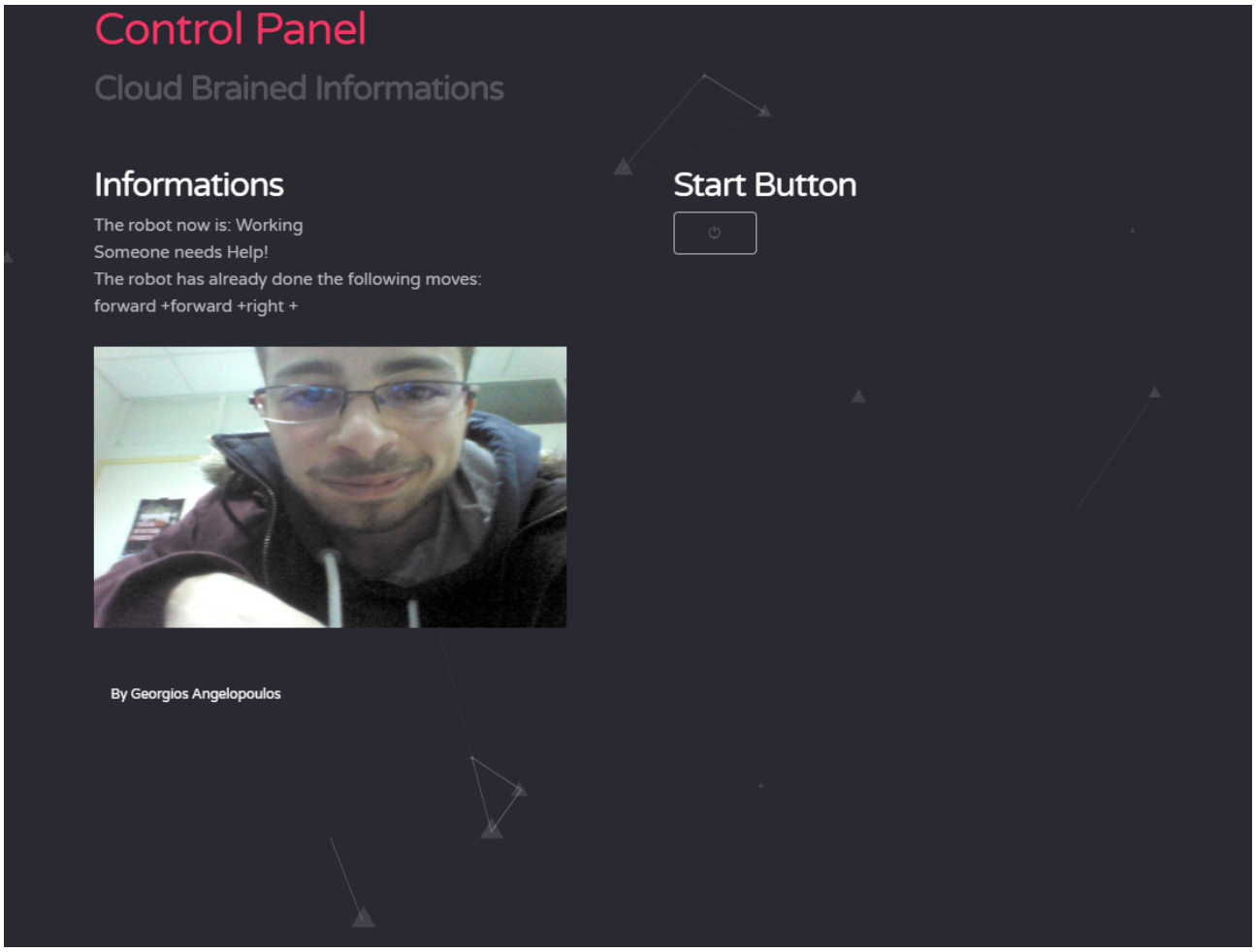


Figure 3.12: Control Panel.

The control panel consists of information collected from the hazardous zone. Specifically,

- The user can view the running mode of the robot,
- The user can view if someone needs help,
- The user can view what moves the robot has already done,
- The user can enable or disable the robot

In order this information to appear in the control panel we used PHP as you can see in the Listing 3.14.

fopen() function was utilized to open files and read the wanted facts. This function gives you more options than the readfile() function. The first parameter of fopen() contains the name of the file to be opened and the second parameter specifies in which mode the file should be opened.

Listing 3.14: Inserting information using PHP.

```
<?php

        $file = "start.txt";
        $handle = fopen($file , "r");
        $contents = fread($handle, filesize($file ));
        fclose($handle);
        fclose($file);
        $r = (1 == $contents ) ? 'Working' : 'Not Working';

        $file = "person.txt";
        $handle = fopen($file , "r");
        $contents = fread($handle, filesize($file ));
        fclose($handle);
        fclose($file);

        if (0 == $contents)
                $d = 'We have not found someone';
        if (1 == $contents)
                $d = 'Someone needs Help!';
        if (2 == $contents)
                $d = 'We have found someone';



        $myfile = fopen("moves.txt", "r");
        $f = fread($myfile,filesize("moves.txt"));
```

```
        fclose($myfile);


?>
```

3.10 Chapter Summary

In chapter 3 we talked about our system architecture some choices regarding movement for the robot was explained. A large part of chapter 3 was devoted to obstacle detection. Also, we talked about important functions that our robot uses. Finally, we presented how the cloud server and the website works.

Chapter 4

Discussion

4.1 Nao and Hazardous Environments

4.1.1 Robot's design limitations

The design limitations of the Nao robot doesn't allow us to perform in every situation. Specifically,

- We can't expose Nao to any form of water as permanent damage may occur. In particular, if Nao switches from a cold environment to a warm one, condensation may occur on its surface or inside.
- Nao is designed to function between 10 and 35 °C (50-95 F).
- Nao is designed to function within 10% to 90% relative humidity.
- Nao will move about properly if the floor is flat, hard and smooth.

4.1.2 Hazardous environments

An environmental hazard is a substance, a state or an event which has the potential to threaten people's health, including pollution and natural disasters such as storms and earthquakes.

Any single or combination of toxic chemical, biological, or physical agents in the environment, resulting from human activities or natural processes, that may impact the health of exposed subjects, including pollutants such as heavy metals, pesticides, biological contaminants, toxic waste, industrial and home chemicals [36].

Hazards can be categorized into four types:

- Chemical
- Physical.
- Biological.
- Psychosocial .

Chemical

A chemical hazard is a type of occupational hazard caused by exposure to chemicals in the workplace. Exposure to chemicals in the workplace can cause acute or long-term detrimental health effects.

Physical

A physical hazard is a type of occupational hazard that involves environmental hazards that can cause harm with or without contact.

Biological

Biological hazards, also known as biohazards, refer to biological substances that pose a threat to the health of living organisms, primarily that of humans. This can include medical waste or samples of a microorganism, virus or toxin (from a biological source) that can affect human health.

Psychosocial

Psychosocial hazards include but aren't limited to stress, violence and other workplace stressors.

4.1.3 Nao on Hazardous environments

As we mention in the sections above due to Nao's design, our humanoid robot cannot perform in every situation. But, can perform with ease to a biological or chemical or in some cases physical hazard.

For example, in a Poor Air Quality environment. Air does not necessarily have to be contaminated to be dangerous. Low oxygen or high dust levels can be fatal.

Apart from that, Nao can perform on Extreme Decibel Levels environment. Prolonged exposure to excessive noise can damage or destroy hearing capabilities, or cause even more serious problems.

4.2 Security

4.2.1 The experiment

To explore the safety of our system, we performed an experiment where a penetration tester tries to hijack our robot.

From this experiment, we discovered some security holes in the system. Firstly, the penetration tester cracked our Wifi password using AirSnort.

AirSnort is another popular tool for decrypting WEP encryption on a wi-fi 802.11b network. It is a free tool and comes with Linux and Windows platforms. This tool is no longer maintained, but it is still available to download from Sourceforge. AirSnort works by passively monitoring transmissions and computing encryption keys once it has enough packets received.

Then, using the technique man-in-the-middle attacking [37], detects the IP of our robot. Finally, the penetration tester connects to the robot and disables it.

4.2.2 Prevention

Having a strong encryption mechanism on wireless access points but also a strong password to our humanoid robot prevents unwanted users from joining our network just by being nearby. A weak encryption mechanism can allow an attacker to brute-force his way into a network and begin man-in-the-middle attacking. The stronger the encryption implementation, the safer.

4.3 SWOT Analysis

SWOT Analysis is a useful technique for understanding the Strengths and Weaknesses, and for identifying both the Opportunities open to us and the Threats that our system faces.

Strengths

- Provides a shared knowledge therefore we can make the robots smarter.
- Offloads heavy computing tasks to the cloud.
- Cheaper, lighter and easy to maintain hardware.
- Longer battery life.

Weaknesses

- The robot is totally dependent on cloud a small fault in network can leave the robot brainless.
- It can be hacked.

Opportunities

- Helps to cut the accidents and health issues by removing workers from hazard and dangerous environments.

Threats

- If our system robot is hacked, victims could have risk of their personal privacy and security. It may be accessed and leaked to the world around by criminals. Another problems is once a robot is hacked and controlled by someone else, which may put the user in danger.

4.4 System Metrics

The system consists of both the server and the robot. For presenting the capabilities and its quality, we run a set of measurements. These measurements concern both parts of the system. More specifically, the main and most important source code of our system from the server side is ”server1.py” Server1.py has:

- Lines of code:358
- Source lines of code:231
- Commented lines of code:47

Apart from that, our server has 6 xml files and 1 folder for our website.

Also, the main and most important source code of our system from the robot side is ”client1.py” Client1.py has:

- Lines of code:428
- Source lines of code:269
- Commented lines of code:64

Finally, the overall software has been rated with A from the Codacy scoring service (Figure 4.1). Our system in general has 118113 Lines of code as calculated from Codacy.

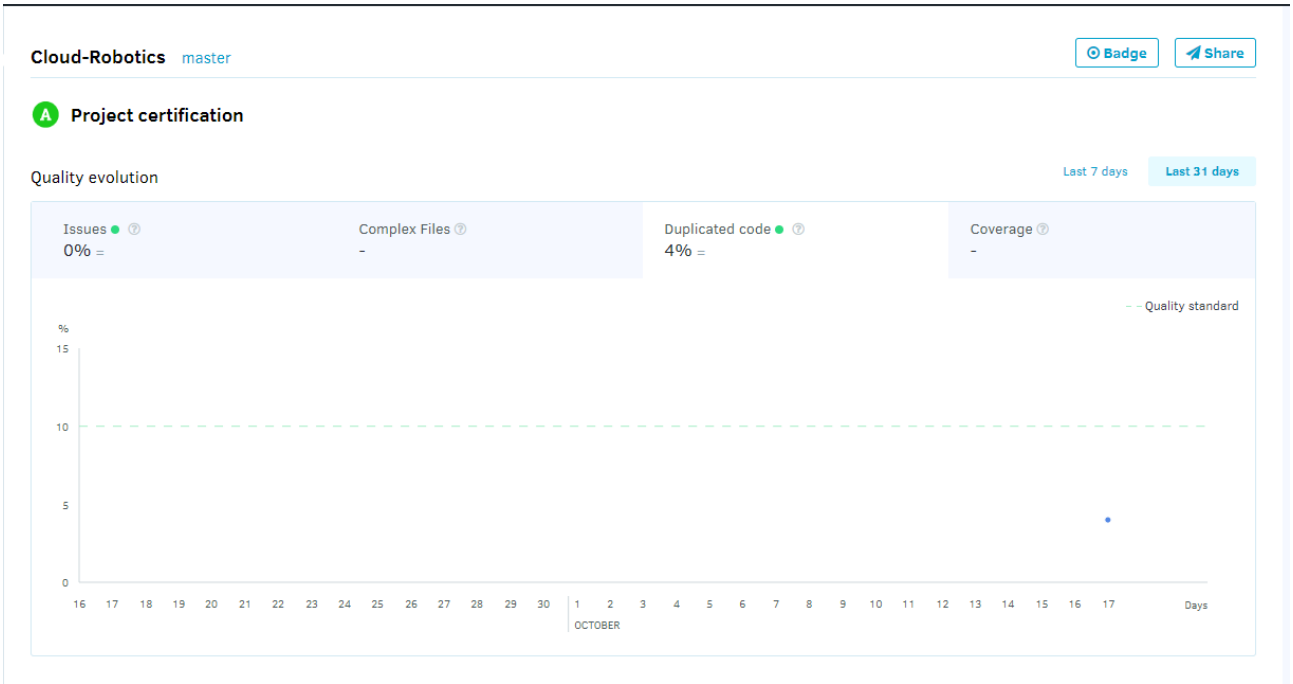


Figure 4.1: Score from Codacy.

4.5 Chapter Summary

At the beginning of chapter 4, we were taking a closer look at how our humanoid robot can work on extreme conditions. Apart from that, we discussed the experiment we performed where a penetration tester tries to hijack our robot. Last but not least, we presented our SWOT analysis.

Chapter 5

Conclusions and Future Work

5.1 Conclusions

In the first "Matrix" movie, there's a scene where Neo points to a helicopter on a rooftop and asks Trinity, "Can you fly that thing?" Her answer: "Not yet." Then she gets a "pilot program" uploaded to her brain and they fly away.

For us humans, with our non-upgradeable, offline meat brains, the possibility of acquiring new skills by connecting our heads to a computer network is still science fiction. Not so for robots.

This thesis proposes a novel, fast and secure system using a cloud server as a brain which communicates with minimal latency with our humanoid robot in order to be able to autonomously move and detect lives in emergency scenarios with the potential to communicate with the victim.

5.2 Results

As an initial test, we created a simulated disaster environment in a 27 square meter room (Figure 5.1). We recruited fifteen volunteers to each perform an evacuation scenario. Our results indicated that all victims will follow the instructions and answer all the questions that our robot asks. However, this simulation was low stress and simple, so in real hazardous environments, victims may react differently. In real disasters, many people will risk personal injury and even death to find members of their group [38]. After the test was finished we received feedback from our volunteers about the experiment. We asked 3 important questions about their emotions when the robot came (Figure 5.2), if the robot was slow during the search and rescue

mission (Figure 5.3) and finally if the environment achieved to simulate a hazardous environment (Figure 5.4)

As you can see in Figure 5.1 our robot with this obstacle ordinance communicates encrypted with our system 13 times in order to decide how to move in the room.

- Point A: Detects obstacle in front and right so with the help of the cloud turns left.
- Point B: Detects obstacle in front so with the help of the cloud turns right.
- Point C: Detects obstacle in front and right so with the help of the cloud turns left.
- Point D: Detects obstacle in front and right so with the help of the cloud turns left.
- Point E: Detects obstacle in front so with the help of the cloud turns right.
- Point F: Detects obstacle in front and left so with the help of the cloud turns right.
- Point G: Detects obstacle in front, left and right so with the help of the cloud turns backwards.
- Point H: Detects obstacle in front and right so with the help of the cloud turns left.
- Point I: Detects obstacle in front and right so with the help of the cloud turns left.
- Point J: Detects obstacle in front and right so with the help of the cloud turns left.
- Point K: Detects obstacle in front left so with the help of the cloud moves right.
- Point L: Detects obstacle in front so with the help of the cloud turns right.
- Point M: Detects victim and asks for help.

In this particular environment, our system detects the victim in an average time of 1 min and 48 sec.

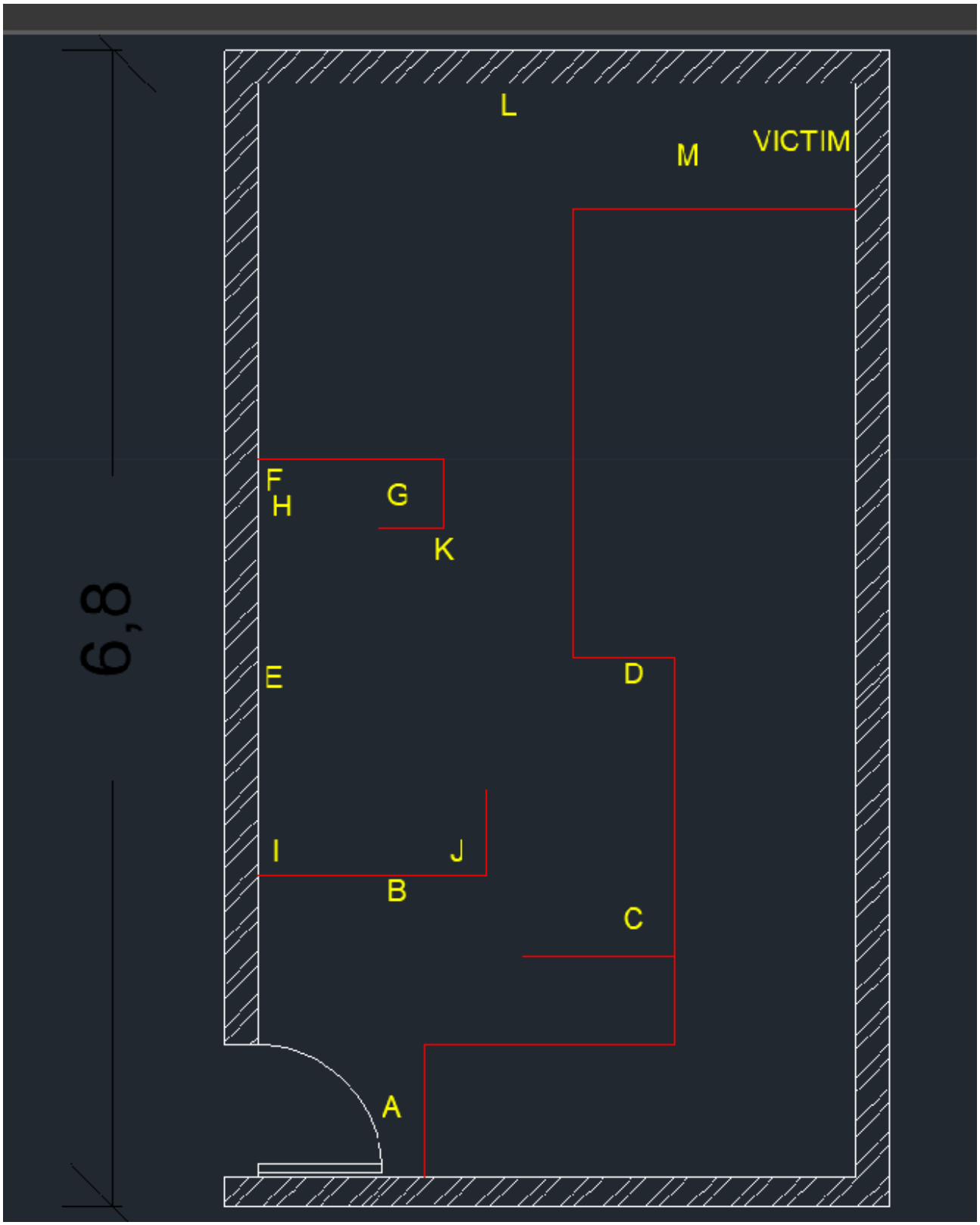


Figure 5.1: The room we used for experiments.

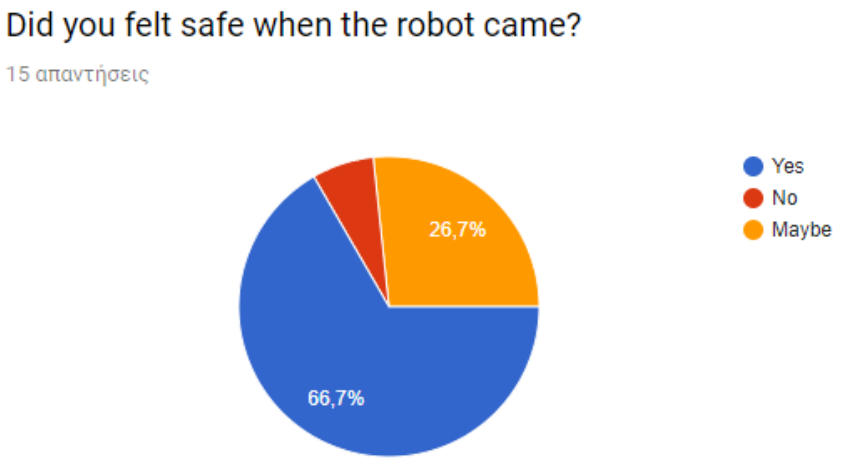


Figure 5.2: 66.7% answered that they felt safe when the robot came.

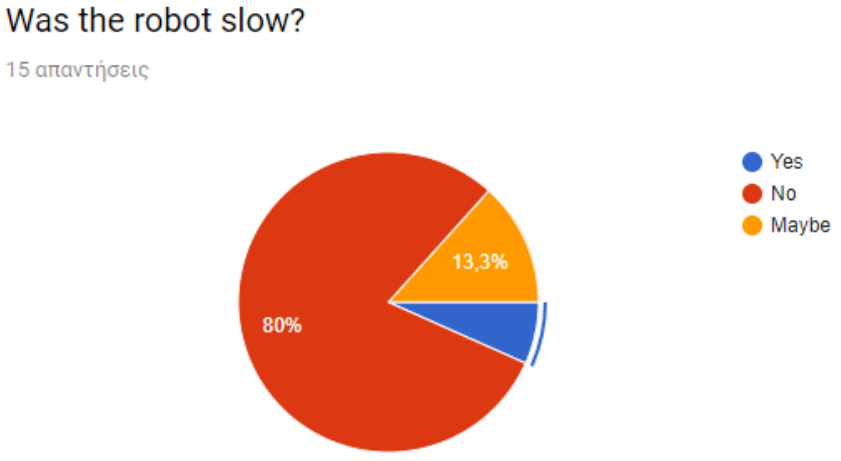


Figure 5.3: 6.7% answered that the robot was slow.

5.3 Future Work

Considering the satisfying results that this study produced, in the future, we plan to create an architecture for our system which leverages the combination of an ad-hoc cloud formed by machine-to-machine (M2M) communications among participating

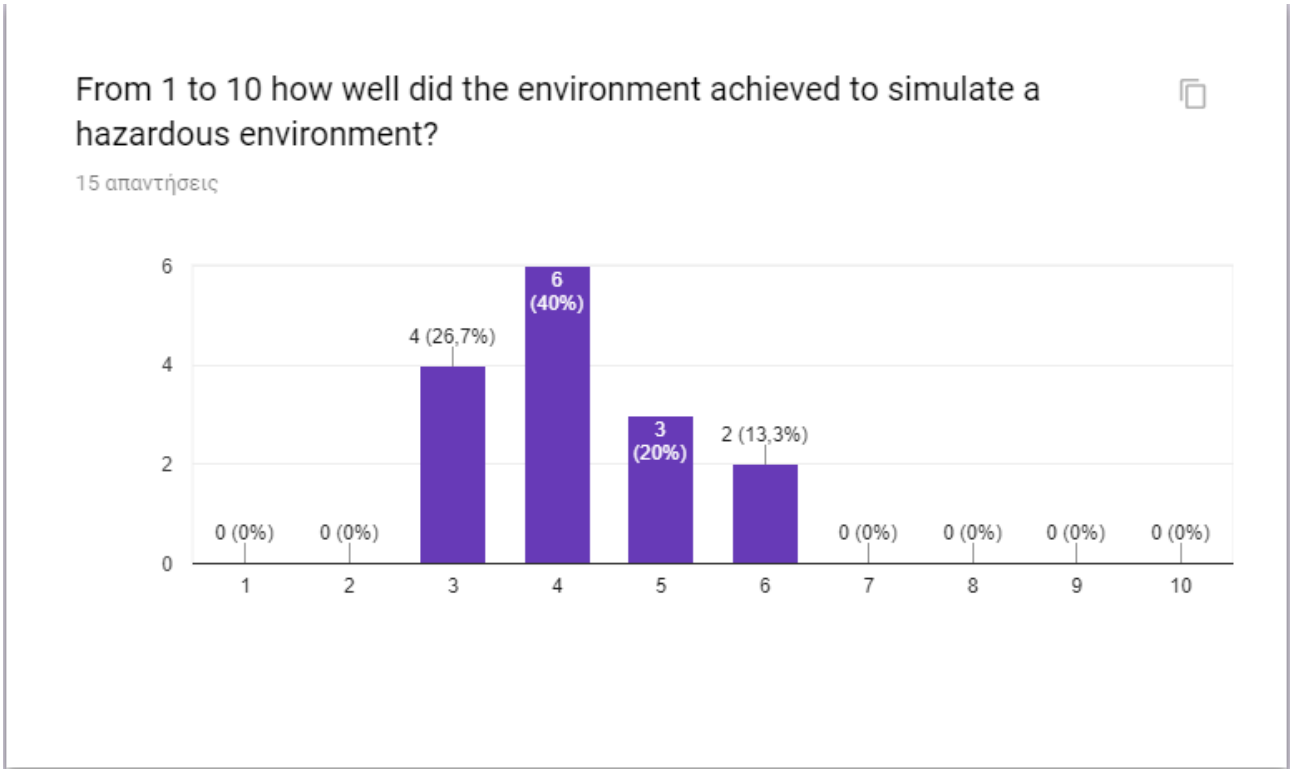


Figure 5.4: As shown from the answers this simulation was low stress and simple.

robots with an infrastructure cloud enabled by machine-to-cloud (M2C) communications .

Also, we plan to enhance the system by inserting more functionalities such as AI.

Appendices

Appendix A

Cloud-Brained procedure

The cloud brained procedure in python is shown in Listing A.1

Listing A.1: Cloud brained procedure.

```
if data == 'Ready':
    file=text(file)
    while (file != '1'):
            file=text(file)
    s.send(encrypt("Begin"))
if data != 'Ready':
    t = open("/home/user/public_html/person.txt", "w+")
    t.write('0')
    t.close()
    part1 = data.split('plus')[0]

#==========================
    if (part1=='object_f'):
            Left = data.split('plus')[1]
            Right = data.split('plus')[2]
            Left=float(Left)
            Right=float(Right)
            if ((Left) > 0.4) and (( Right) > 0.4):
                print("Going Forward")
            all_moves= 'forward +' + all_moves
```

```
            s.send(encrypt("Forward"))
        else:
            print("Use Camera")
            s.send(encrypt("Camera"))

#==========================

    if (part1=='camera'):
        serv = socket.socket(socket.AF_INET, socket.SOCK_STREAM
           )
                serv.bind(server_address1)
                serv.listen(5)
                print 'listening ...'
                while True:
                        conn, addr= serv.accept()
                        print 'client connected ... ', addr
                        myfile = open('image.jpg', 'w')
                        while True:
                data = conn.recv(4096)
                                if not data: break
                                myfile.write(data)
                                print 'writing file ....'
                        myfile.close()
                        print 'finished writing file'
                        conn.close()
                        print 'client disconnected'
                        file1 = open('image.jpg', 'r')
                        file=file1.read()
                        file1.close()
                        data=file
                        # DECRYPT: AES 128 bit, CBC
                        obj2 = AES.new(keySizeInBits128, AES.
```

```
                    MODE_CBC, 'This is an IV456')
                data = obj2.decrypt(data)
                data = data.split('end')[0]
                # DECRYPT done
                string=data
                t = open("image.jpg", "wb+")
                t.write(string)
                t.close()
                break
        j = open("public_html/panel/moves.txt", "w+")
        j.write(all_moves)
        j.close()
        t = open("public_html/image_upload/uploads/
           image.jpg", "wb+")
        t.write(string)
        t.close()
        print("Processing Image")
        face_cascade = cv2.CascadeClassifier('
           haarcascade_frontalface_default.xml')
        eye_cascade = cv2.CascadeClassifier('
           haarcascade_eye.xml')      # load the input
           image and convert it to grayscale
        image = cv2.imread("image.jpg")
        gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
        # load the  Haar cascade, then detect faces
        # in the input image
        faces = face_cascade.detectMultiScale(gray, 1.3
           , 5)
        cat_detected = False
        for (i,(x,y,w,h)) in enumerate(faces):
            cv2.rectangle(image,(x,y),(x+w,y+h),(255,0,
               0),2)
```

```
                roi_gray = gray[y:y+h, x:x+w]
                roi_color = image[y:y+h, x:x+w]
                eyes = eye_cascade.detectMultiScale(
                   roi_gray)
                for (ex,ey,ew,eh) in eyes:
                    cv2.rectangle(roi_color,(ex,ey),(ex+ew,
                       ey+eh),(0,255,0),2)
                    cv2.putText(image, "Person #{}".format(
                       i + 1), (x, y - 10), cv2.
                       FONT_HERSHEY_SIMPLEX, 0.55, (0, 0, 2
                       55), 2)
                    cat_detected = True
            print  'Face',cat_detected
            if cat_detected == False:
                    detector = cv2.CascadeClassifier('
                       haarcascade_frontalcatface.xml')
                    rects = detector.detectMultiScale(gray,
                        scaleFactor=1.3, minNeighbors=10,
                       minSize=(75, 75))
                    person_detected = False
                    # loop over the cat faces and draw a
                       rectangle surrounding each
                    for (i, (x, y, w, h)) in enumerate(
                       rects):
                            cv2.rectangle(image, (x, y), (x
                                + w, y + h), (0, 0, 255), 2
                               )
                            cv2.putText(image, "Cat #{}".
                               format(i + 1), (x, y - 10),
                               cv2.FONT_HERSHEY_SIMPLEX, 0.
                               55, (0, 0, 255), 2)
                            person_detected = True
```

```
                        # show the detected cat faces
                        print 'Cat',person_detected
                if (cat_detected == True) or (person_detected
                   == True):
                        s.send(encrypt("Voice"))
                if (cat_detected != True) and  (person_detected
                    != True):
                        s.send(encrypt("NoFace"))

#==========================

        if (part1=='object_hand'):
                ar = data.split('plus')[1]
                de = data.split('plus')[2]
                ar=float(ar)
                de=float(de)
                if ar > 1.18 and de < -1.18:
                        if (Left - Right) > 0.3:
                        # PAEI ARISTERA BRHKE EMPODIO MPROSTA
                           DEKSIA
                                all_moves= 'forward_left +' +
                                   all_moves
                                s.send(encrypt("forward_left"))
                        elif (Left - Right) < -0.33:
                        # PAEI ARISTERA BRHKE EMPODIO MPROSTA
                           DEKSIA
                                all_moves= 'forward_right +' +
                                   all_moves
                                s.send(encrypt("forward_right")
                                   )


                        else:
```

```
                                all_moves= 'right +' +
                                   all_moves
                                s.send(encrypt("right"))

                        elif ar > 1.18 and de > -1.18:
                        all_moves= left +' + all_moves
                        s.send(encrypt("left"))
                        elif ar < 1.18 and de < -1.18:
                        all_moves= 'right +' + all_moves
                        s.send(encrypt("right"))

                        else:
                        all_moves= 'back +' + all_moves
                        s.send(encrypt("back"))

#==========================

        if (part1=='voice'):
                serv = socket.socket(socket.AF_INET, socket.
                   SOCK_STREAM)
                serv.bind(server_address1)
                serv.listen(5)
                print 'listening ...'

                while True:
                        conn, addr= serv.accept()
                        print 'client connected ... ', addr
                        myfile = open('voice.wav', 'w')

                        while True:
                                data = conn.recv(4096)
                                if not data: break
```

```
                        myfile.write(data)
                        print 'writing file ....'

                myfile.close()
                print 'finished writing file'
                conn.close()
                print 'client disconnected'
                file1 = open('voice.wav', 'r')
                file=file1.read()
                file1.close()
                data=file
                # DECRYPT: AES 128 bit, CBC
                obj2 = AES.new(keySizeInBits128, AES.
                   MODE_CBC, 'This is an IV456')
                data = obj2.decrypt(data)
                data = data.split('end')[0]
                # DECRYPT done

                        string=data
                t = open("voice.wav", "wb+")
                t.write(string)
                t.close()
                break
        print("Processing Sound")
        r = sr.Recognizer()
        with sr.WavFile("voice.wav") as source:
                        audio = r.record(source)
                        # extract audio data from the
                           file
        try:
                print( r.recognize_google(audio))
                # recognize speech using Google Speech
```

```
                        Recognition
                if "yes" in  r.recognize_google(audio):
                        t = open("/home/user/public_html/panel/
                           person.txt", "w+")
                                t.write('1')
                                t.close()
                else:
                        t = open("/home/user/public_html/panel/
                           person.txt", "w+")
                                t.write('2')
                                t.close()

                except LookupError:
                # speech is unintelligible
                        print("Could not understand audio")
                break

#==========================

    file=text(file)
    if file == '0':
        s.send(encrypt("Stop"))
    if data == '1.Done':
        s.send(encrypt("goto2"))

#==========================
```

Appendix B

Installation Manual

In order to implement the thesis you will need a Nao version 5 and a server ideally with Ubuntu OS.

B.1 Nao

B.1.1 NAOqi OS - user accounts

The main user is nao, and like any GNU/Linux system, there is the super-user root.

By default, passwords are usernames. So, changing user to root using the su command will request the password root

Logging in as root over ssh is now disabled. However the su command remains available.

We recommend to change the nao‘s password using the web page.

B.1.2 Accessing NAO over ssh

To login on your Aldebaran robot, get its IP address pushing its torse button, then connect to your Aldebaran robot over ssh using PuTTy. Afterward, you can install the Nao’s code from the link below:

https://github.com/angelopoulosG/Cloud-Robotics

B.2 Server

Python is crucial for our server in order to run our code.

So first, you have to install some dependencies as shown in Listing B.1.

Listing B.1: Python installation part 1.

```
sudo apt-get install build-essential checkinstall
sudo apt-get install libreadline-gplv2-dev libncursesw5-dev
   libssl-dev libsqlite3-dev tk-dev libgdbm-dev libc6-dev libbz
   2-dev
```

Then download using the following command as shown in Listing B.2.

Listing B.2: Python installation part 2.

```
version=2.7.13
cd ~/Downloads/
wget https://www.python.org/ftp/python/$version/Python-$version
   .tgz
```

Extract and go to the directory as shown in Listing B.3.

Listing B.3: Python installation part 3.

```
Python
tar -xvf Python-$version.tgz
cd Python-$version
```

Now, install using the command you just tried, using checkinstall instead to make it easier to uninstall if needed as shown in Listing B.4.

Listing B.4: Python installation part 4.

```
./configure
make
sudo checkinstall
```

And now you have to install using PIP the following libraries:

- socket
- opencv

- speech_recognition
- Crypto.Cipher
- base64

Finally, you will need to install PHP as shown in Listing B.5.

Listing B.5: PHP installation.

```
sudo apt-get install php
```

All the necessary files and codes for the server are stored in the link below: https://github.com/angelopoulosG/Cloud-Robotics

Bibliography

[1] I. Foster, Y. Zhao, I. Raicu, and S. Lu. Cloud computing and grid computing 360-degree compared. In *2008 Grid Computing Environments Workshop*, pages 1–10, Nov 2008. doi: 10.1109/GCE.2008.4738445.

[2] J. Wan, S. Tang, H. Yan, D. Li, S. Wang, and A. V. Vasilakos. Cloud robotics: Current status and open issues. *IEEE Access*, 4:2797–2807, 2016. doi: 10.1109/ACCESS.2016.2574979.

[3] R. Arumugam, V. R. Enti, L. Bingbing, W. Xiaojun, K. Baskaran, F. F. Kong, A. S. Kumar, K. D. Meng, and G. W. Kit. Davinci: A cloud computing framework for service robots. In *2010 IEEE International Conference on Robotics and Automation*, pages 3084–3089, May 2010. doi: 10.1109/ROBOT.2010.5509469.

[4] Robotics technology thrust. `https://www.energy.gov/sites/prod/files/2016/09/f33/DOE%20NCW%20-%20160915%20FINAL%20REV01.pdf`. [Online; accessed 28 September 2018].

[5] Robot 510 packbot. irobot. `http://www.irobot.com/en/us/learn/defense/packbot.aspx`. [Online; accessed 28 September 2018].

[6] K. Nagatani, S. Kiribayashi, Y. Okada, S. Tadokoro, T. Nishimura, T. Yoshida, E. Koyanagi, and Y. Hada. Redesign of rescue mobile robot quince. In *2011 IEEE International Symposium on Safety, Security, and Rescue Robotics*, pages 13–18, Nov 2011. doi: 10.1109/SSRR.2011.6106794.

[7] Overveiw of the robot□survey runner. `http://www.tepco.co.jp/en/nu/fukushima-np/images/handouts_120417_03-e`. [Online; accessed 28 September 2018].

[8] Ivaldi Anzalone, Boucenna and M. Chetouani. Evaluating the engagement with social robots. *International Journal of Social Robotics*, page 465–478, 2015.

[9] Brian Scassellati, Henny Admoni, and Maja Mataric. Robots for use in autism research. *Annual Review of Biomedical Engineering*, 14(1):275–294, 2012.

[10] Astrid Weiss, Regina Bernhaupt, Michael Lankes, and Manfred Tscheligi. The usus evaluation framework for human-robot interaction. 2009.

[11] B. M. Muir and N. Morayi. Trust in automation. part ii. experimental studies of trust and human intervention in a process control simulation. In *Ergonomics*, volume 39, 1996.

[12] M. Mundy S. Lewandowsky and G. Tan. The dynamics of trust: comparing humans to automation. In *Journal of Experimental Psychology: Applied*, volume 6, 2000.

[13] H. Funkhouser E. Lyman C. Billings, J. Lauber and E. Huff. Nasa aviation safety reporting system. In *Tech. Rep. Technical Report TM-X-344*, 1976.

[14] C. Drury Y. Seong J. Llinas, A. Bisantz and J.-Y. Jian. Studies and analyses of aided adversarial decision making. phase 2: Research on human trust in automation. In *DTIC Document, Tech. Rep.*, 1998.

[15] J. D. Lee and K. A. See. Trust in automation: Designing for appropriate reliance. In *Human Factors: The Journal of the Human Factors and Ergonomics Society*, volume 46, 2004.

[16] G. Angelopoulos, G. T. Kalampokis, and M. Dasygenis. An internet of things humanoid robot teleoperated by an open source android application. In *2017 Panhellenic Conference on Electronics and Telecommunications (PACET)*, pages 1–4, Nov 2017. doi: 10.1109/PACET.2017.8259978.

[17] K. Hirai, M. Hirose, Y. Haikawa, and T. Takenaka. The development of honda humanoid robot. In *Proceedings. 1998 IEEE International Conference on Robotics and Automation (Cat. No.98CH36146)*, volume 2, pages 1321–1326 vol.2, May 1998. doi: 10.1109/ROBOT.1998.677288.

[18] Y. Sakagami, R. Watanabe, C. Aoyama, S. Matsunaga, N. Higaki, and K. Fujimura. The intelligent asimo: system overview and integration. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 3, pages 2478–2483 vol.3, Sept 2002. doi: 10.1109/IRDS.2002.1041641.

[19] K. Kaneko, S. Kajita, F. Kanehiro, K. Yokoi, K. Fujiwara, H. Hirukawa, T. Kawasaki, M. Hirata, and T. Isozumi. Design of advanced leg module for humanoid robotics project of meti. In *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No.02CH37292)*, volume 1, pages 38–45 vol.1, May 2002. doi: 10.1109/ROBOT.2002.1013336.

[20] N. Kanehira, T. U. Kawasaki, S. Ohta, T. Ismumi, T. Kawada, F. Kanehiro, S. Kajita, and K. Kaneko. Design and experiments of advanced leg module (hrp-2l) for humanoid robot (hrp-2) development. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 3, pages 2455–2460 vol.3, Sept 2002. doi: 10.1109/IRDS.2002.1041636.

[21] K. Kaneko, F. Kanehiro, S. Kajita, H. Hirukawa, T. Kawasaki, M. Hirata, K. Akachi, and T. Isozumi. Humanoid robot hrp-2. In *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004*, volume 2, pages 1083–1090 Vol.2, April 2004. doi: 10.1109/ROBOT.2004.1307969.

[22] K. Akachi, K. Kaneko, N. Kanehira, S. Ota, G. Miyamori, M. Hirata, S. Kajita, and F. Kanehiro. Development of humanoid robot hrp-3p. In *5th IEEE-RAS International Conference on Humanoid Robots, 2005.*, pages 50–55, Dec 2005. doi: 10.1109/ICHR.2005.1573544.

[23] About okeanos. https://~okeanos.grnet.gr/about/. [Online; accessed 28 September 2018].

[24] Dave. Kuhlman. *A Python Book: Beginning Python, Advanced Python, and Python Exercises*.

[25] What is css? https://www.w3.org/standards/webdesign/htmlcss#whatcss. [Online; accessed 28 September 2018].

[26] Flanagan David. *JavaScript - The definitive guide*. O'Reilly Media, 2008.

[27] History of php. http://php.net/, . [Online; accessed 28 September 2018].

[28] Introduction: What can php do? https://secure.php.net/manual/en/intro-whatcando.php, . [Online; accessed 28 September 2018].

[29] Behrouz A. Forouza. *Cryptography and Network Security*. McGraw Hill publication, 2010.

[30] *Advanced Encryption Standard*. Federal Information Processing Standards Publication 197, 2001.

[31] William Stallings. *Cryptography and Network Security Principles and Practice*. Pearson publications, 2006.

[32] Joan Daemen and Vincent Rijman. *AES proposal document*. 1999.

[33] Lynn Hathaway. *National Policy on the Use of the Advanced Encryption Standard (AES) to Protect National Security Systems and National Security Information*. 2003.

[34] Opencv software overview. http://opencv.org/about.html. [Online; accessed 28 September 2018].

[35] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, volume 1, pages I–I, Dec 2001. doi: 10.1109/CVPR.2001.990517.

[36] Environmental hazard. https://definedterm.com/environmental_hazard. [Online; accessed 28 September 2018].

[37] Man in the middle. https://github.com/jtesta/ssh-mitm. [Online; accessed 28 September 2018].

[38] J. D Sime. *Affiliate behaviour during escape to building exits*. Journal of Environmental Psychology, 1983.