Università degli Studi dell'Aquila

Engineering

Cognitive Robotics Systems Engineering: A Pilot Project of an Empathic Robot

Author:
Agnese Salutari

Supervisor:
Prof. Giovanni De Gasperis

Co-Supervisor:
Prof. Laura Tarantino

A thesis submitted for the degree of

Ingegneria Informatica e Automatica

March 27, 2020

Abstract

Something important is happening in our everyday life: robots and smart technology in general are becoming more and more present in many different aspects, in ways we couldn't even imagine some years ago.

Human-machine interaction is becoming a fundamental part in systems design, so we have now to face with new aspects of robotics engineering. We need robots to be smart in order to understand user needs deeply, in order to be proactive, acting in the proper way in every contest. Here comes the necessity for robots cognition abilities improvement.

In this work I'm going to talk about Empathic Robotics, that is the nascent engineering discipline that studies and facilitate human-robot interactions and optimizes robots actions and reactions, making them appropriate to user feelings and needs. In particular, I'm going to analyze the highest form of user requests detection, feelings recognition. At the same time, my aim is to present practical examples of technologies that can be used to make these kind of systems, showing some little projects I performed during my research. These projects are modular and can be used independently or together to expand pre-existing systems or to create new ones. Finally, I'll present a real pilot project of such an empathic robot, from design to final implementation, going through developing.

«"O frati," dissi, "che per cento milia
perigli siete giunti a l'occidente,
a questa tanto picciola vigilia

d'i nostri sensi ch'è del rimanente
non vogliate negar l'esperïenza,
di retro al sol, del mondo sanza gente.

Considerate la vostra semenza:
fatti non foste a viver come bruti,
ma per seguir virtute e canoscenza".»

(Dante Alighieri, Inferno, XXVI 112-120)

Acknowledgements

English

I am a lucky person.

I was born in the land that gave birth to the Marsi warriors, and I was lucky enough to study in a city whose name, L'Aquila, means "eagle", but that is more like a phoenix, because it is being reborn from its own ashes.

Furthermore I'm doing what I love the most, programming, and I am realizing my dream as a child, working with robots.

Anyway, my biggest luck is people that surrounds me, and that I would like to thank. In doing so, I will proceed in chronological order, because every person I'm going to thank is very important to me and I cannot choose who could be the first one.

Thanks to the very first persons I met, my parents, Antonella and Stefano, and my family, Maria, Antonio, Agnese, Felicita, Alberto, Costanza, Marco, Vittorio, Valerio, and every cousin, aunt and uncle (you are a lot!). You are my family, my best friends and my encouragement team. This thesis is dedicated to you.

I would like to thank Università degli Studi dell'Aquila Professors and Staff for having helped me to grow professionally and as a person. I want to thank especially two of them, Professor Giovanni De Gasperis and Professor Laura Tarantino.

Like all the best things, I met Professor De Gasperis at an unexpected moment, during the three-year degree. He is always working on a hundred projects in parallel at the same time, and he always solves every problem... Actually I suspect he can travel through time. He taught me almost everything I know about robots and I will never stop thanking him.

I met Professor Tarantino for the first time during the three-year degree, but I started knowing her better during my master degree. She has the super power of making the most difficult things seem simple (until you try doing these same things by yourself!) and nothing can stop her. I had the luck of working with her during PinKamp 2019.

However, another Professor Tarantino and Professor De Gasperis regard I have to praise is their patience with me. Thank you again!

Finally thanks to my friends for their nice company during all these years.

And thanks to you too, that are reading. If you arrived at this point of Acknowledgements, I'm also lucky because you are a gentle reader.

Italiano

Sono una persona fortunata.

Sono nata nella terra che ha dato i natali ai guerrieri Marsi, e sono stata così fortunata da studiare nella città il cui nome è "L'Aquila", ma che è più simile ad una fenice perché sta rinascendo dalle proprie ceneri.

Inoltre faccio quello che amo di più, programmo, e sto realizzando il sogno della mia infanzia, lavoro con i robot.

Ad ogni modo, la mia più grande fortuna è la gente che mi circonda e che vorrei ringraziare. Nel farlo, procederò in ordine cronologico, perché ogni persona che sto per ringraziare è molto importante per me e non saprei scegliere chi dovrebbe essere il primo.

Grazie alle prime persone in assoluto che ho incontrato, i miei genitori, Antonella e Stefano, e la mia famiglia, Maria, Antonio, Agnese, Felicita, Alberto, Costanza, Marco, Vittorio, Valerio, e ogni cugino, zia e zio (siete proprio tanti!). Voi siete la mia famiglia, i miei migliori amici e la mia equipe d'incoraggiamento. Questa tesi è dedicata a voi.

Vorrei ringraziare i Professori e il Personale dell'Università degli Studi dell'Aquila per avermi aiutata a crescere professionalmente e come persona. Voglio ringraziare soprattutto due di loro, il Professor Giovanni De Gasperis e la Professoressa Laura Tarantino.

Come tutte le cose migliori, ho incontrato il Professor De Gasperis in un momento inaspettato, durante la laurea triennale. Lui lavora sempre su centinaia di progetti in parallelo nello stesso tempo e risolve ogni volta tutti i problemi... In verità sospetto che sappia viaggiare nel tempo. Mi ha insegnato quasi tutto quello che so sui robot e non finirò mai di ringraziarlo.
La prima volta ho incontrato la Professoressa Tarantino durante la laurea triennale, ma ho cominciato a conoscerla meglio durante la magistrale. Ha il super potere di far sembrare semplici le cose più difficili (finché non provi a farle da solo!) e niente può fermarla. Ho avuto la fortuna di lavorare con lei in occasione del PinKamp 2019.
Devo però elogiare un'altra qualità della Professoressa Tarantino e del Professor De Gasperis, ovvero la loro pazienza con me. Grazie ancora!

Infine, grazie ai miei amici per avermi fatto compagnia in tutti questi anni.

E grazie anche a te, che stai leggendo. Se sei arrivato fino a questo punto dei ringraziamenti, sono fortunata anche per il fatto che sei un gentil lettore.

Figure 1: Professor De Gasperis and me at my three-year graduation

Figure 2: Professor Tarantino and me at PinKamp 2019

Figure 3: My family and me

Contents

List of Figures

Introduction

Judging is part of human nature. People have an opinion about everything, even about things or other people they don't know. Essence is obviously a must, but the very first thing we can perceive is appearance, so our judging process starts from aesthetics. The next thing we can notice by watching at something or someone is its ability to convey emotions.

When we interact with other persons, our feelings about them are often given by their attitudes and by the way the do things: the tone of their voice, their prettiness and so on. So what we say or what we do are not all the information we communicate each other, but a lot of information is made by how we do that. In particular, we think someone is a good interlocutor when he is able to express his emotions and point of view by corroborating his speech with a proper attitude, and we sympathize with him if he considers our feelings and properly reacts while talking with us.

Objects and even concepts can inspire emotions too, in effect everyone has a favourite color, a favourite pair of shoes, etc. and if we are asked to tell the reason for that preference we almost always are not able to explain it with logical causes. We often anthropomorphize things.

Turing Test has been developed in 1950 by Alan Turing and is aimed to test if a machine could be perceived like a person by a human observer. In its traditional form, it provides that a computer and a person are hidden from the human observer, who has to question both in order to determine which of the two is the machine.

Nowadays machines have to work together with people, there is no more a "curtain" to hide them, so they can not ignore the problem of being included in human society. Machines have now to face with the rules of human society.

The term "robot" comes from the Czech word "ròbota", that means "to hardly work", describing something that should do strenuous things instead of man. We found it for the first time in 1920, used by Czech playwright Karel Čapek for his 1920 play entitled "R.U.R.", that obviously belongs to Science Fiction genre.

Technology has evolved over time, making robots a concrete reality. At the same time, our idea about robots was changing over time, and it continues changing nowadays too. An aspect that has always fascinated us, almost as a common factor, is the relationship between human and robot. In particular, it is spontaneous to ask how emotions, that distinguishes man from machines, could be "perceived" or treated by a robot.

At this point, a natural consequence is questioning about robot ability to "experience" feeling too, or at least to simulate them. Clearly, that emotion simulation is not useful for the robot itself, but it is aimed to satisfy human needs: a person expects a certain emotional feedback by his interlocutor and feels comfortable when he has to do with someone who is in a similar mood to his. That leads to a deep study about human-machine interactions, whom Robotic Engineering can't be indifferent to.

For this thesis I created BlocksBot, the implementation of an Empathic Robot for both simulation and real world environments. To do this, I created my own tools, that are mostly frameworks and libraries, and I used different programming techniques. I finally performed tests with simulations and real robot. The result is the Distributed Hybrid System that I am going to describe.

First of all, I will introduce Empathic Robotics, going through Human-Machine interaction problem analysis and examples. Then, I will present my tools. Finally, I will talk in details about my project.

The remainder of this thesis is structured in the following way:

- Chapter 1:

  In this chapter I am going to talk about human-robot interactions. I will introduce some examples of machines working with people and their main classification: realistic robots VS stylized robots. Then I will talk a little about a particular kind of human-machine interaction: Chatterbots. Delicate aspects, like interactions with children and elderly persons and ethics, will also be briefly discussed.

- Chapter 2:

  In this chapter I will describe hybrid approach and distributed approach for programming robots, providing its features and converging in Distributed hybrid Applications. I will describe my robotic projects, that can be plugged together or in pre-existent systems to easily solve common robotics problems. They are libraries and frameworks to build distributed hybrid systems where logic programming modules, MASs, Neural Networks and procedural/Object Oriented modules can eventually be plugged and work together easily, both in simulation and real world environments. There are a library for managing drones and an hardware/software project for robots that have to work in areas where Internet connection is not available. Their code is available on GitHub (links will be provided)

- Chapter 3:

  In this chapter I will present and describe BlocksBot, my Empathic Robot project. First of all, I will present technologies I used to realize it, that are the message broker, that is Redis, the image and audio processing tools, that are Face++ and Vokaturi, and the real robot I used, Nao. Then I will present my project module by module in details. Code is available on GitHub at the given link.

- Chapter 4:

  In this final chapter, I am going to describe what's next for nascent BlocksBot project.

- Appendix A:

  In Appendix A you can find all sitography of the State-of-the-art, described in Chapter 1.

- Appendix B:

  In Appendix B you can find all the useful links related to my projects.

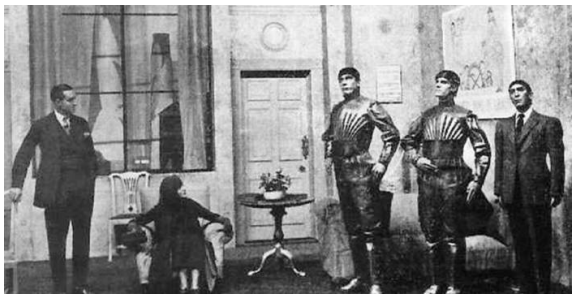


Figure 4: R.U.R. scene

Chapter 1

Robots interacting with people: a state of the art

In this chapter I am going to talk about human-robot interactions. I will introduce some examples of machines working with people and their main classification: realistic robots VS stylized robots. Then I will talk a little about a particular kind of human-machine interaction: Chatterbots. Delicate aspects, like interactions with children and elderly persons and ethics, will also be briefly discussed.

Has a robot to look like a human to better work with people? Actually, many of the robots that have the task to interact with people are humanoid, and some of there robots are as similar as possible to humans. According to their manufacturers, that characteristic helps the robot to be appreciated by users, especially while working in social contexts.

There are an increasing number of applications in which robot work consists in helping people and interacting with them as kindly as possible, so robot appearance and mood make up an important part of requirements. The way the user considers the robot is responsible for robot acceptance indeed, and it can make the difference, especially when the user is a child or an elderly person.

There are two main policies for user-friendly robot design:

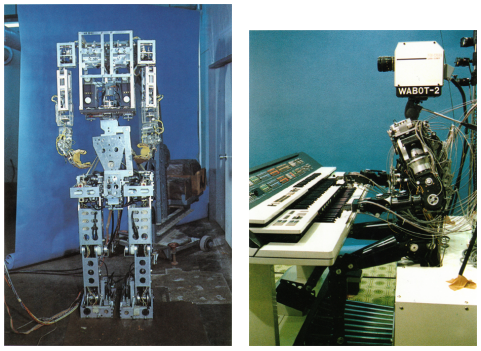
- Humanoid robots with a very realistic body
- Puppet robots with a stylized body

Often, realistic robots tend to scary some users and puppet robots are more appreciated instead, because of their being cute and funny. Puppet robots are similar to cartoons characters, they can inspire good feelings to people by being pretty and are able to show emotions with the same effectiveness of their realistic competitors. In effect, looking at some recent studies, [1] for example), the thesis that realistic robots are more liked by people is not correct. People tend to judge realistic robots more severely, as if they were other people: a more human aspect creates higher expectations. Puppet robots instead are appreciated for their cuteness and have proven to be at least as effective as their competitors in many different tests.

1.1 The WABOT Project

Four laboratories in the School of Science & Engineering of Waseda University joined to set up "The Bio-engineering group" which started the WABOT project in 1970. "WABOT" is an acronym for "WAseda roBOT" (see [1]). The WABOT-1 has been developed between 1970 and 1973 and it is the first fun-scale anthropomorphic robot developed in the world. It consisted of a limb-control system, a vision system and a conversation system. The WABOT-1 is able to communicate-with a person in Japanese and to measure distances and directions to the objects using external receptors, artificial ears and eyes, and an artificial mouth. The WABOT-1 can walk with his lower limbs and is able to grip and transport objects with hands, using tactile-sensors. It has the mental faculty of a one-and-half-year-old child.

In 1980 WABOT-2 project was started. The WABOT-2 is able to play music, but it is not as versatile as its predecessor. This robot musician is able to converse with a person, read a normal musical score with its eye and play tunes of average difficulty on an electronic organ. The WABOT-2 is also able of accompanying a person while he listens to the person singing. The WABOT-2 is the first milestone in developing a "personal robot".



(a) The WABOT-1 (b) The WABOT-2

Figure 1.1: The WABOT Project

1.2 Kismet

Kismet (see [2]) is a robotic head developed by Dr. Cynthia Breazeal at Massachusetts Institute of Technology in the late 1990s. That robot is the representation of a human head, as you can see in the figure Fig 1.2.

It is provided of 2 cameras, hidden behind its eyes, and auditory and proprioceptive sensors, that constitute its inputs. Conversely, its actuators are a speech synthesizer and motors used to move its face, in order to show different emotions.

Facial expressions are created through movements of the ears, eyebrows, eyelids, lips, jaw, and head. The speech synthesizer can simulate personality during speech.

Kismet software is made of different modules, called Systems, as you can see in figure Fig 1.3. It elaborates perceptions based on the input coming from sensors. Perceptions are involved in planning by the Attention, the Behavioural and the Motivation modules. The Attention module is in charge to assign priority to different behaviours, determining the current goal, that is the behaviour with the highest score. To perform the goal, the robot has to properly use its actuators, in order to show the emotion related to the actual behaviour.

[1] http://www.humanoid.waseda.ac.jp/booklet/kato_2.html
[2] http://www.ai.mit.edu/projects/humanoid-robotics-group/kismet/kismet.html

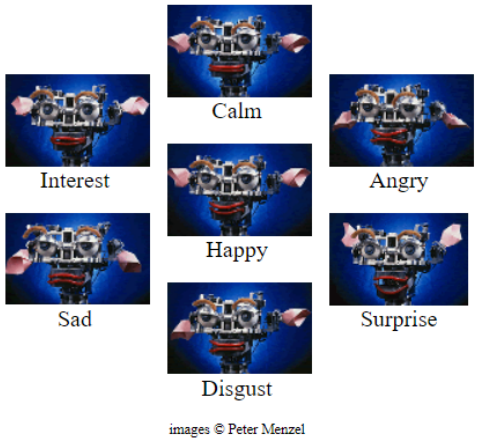


Figure 1.2: Kismet showming different emotions

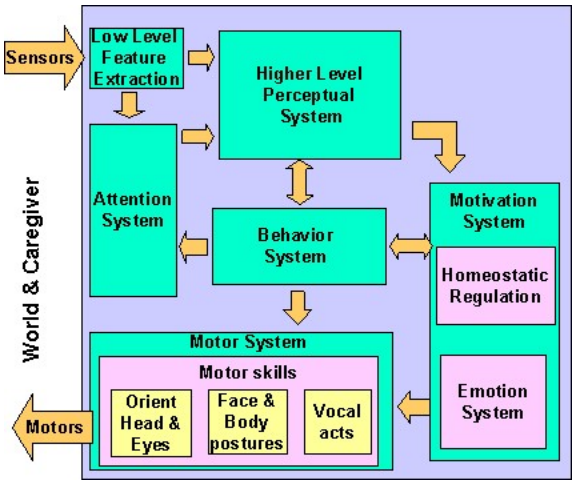


Figure 1.3: Kismet design

There are similar studies performed on robot heads or busts, like:

- MEXI (see [2]) that is an acronym for "Machine with Emotionally eXtended Intelligence
- Emotion Expression Humanoid Robot (see [3])
- ROMAN (see [3])
- Flobi (see [4])
- Kaspar (see [5])
- Furhat (see [6])
- Actdorid-SIT (see [4])
- GolemX-1 (see [5])
- Picoh (see [7])

[3] http://www.takanishi.mech.waseda.ac.jp/top/research/we/we-4rII/index.htm
[4] http://aiweb.techfak.uni-bielefeld.de/flobi-head?utm_source=robots.ieee.org
[5] https://robots.ieee.org/robots/kaspar/?gallery=photo4
[6] https://www.furhatrobotics.com/
[7] https://www.kickstarter.com/projects/ohbot2/picoh-an-expressive-little-robot-head

1.3 ASIMO

ASIMO (see [8]) is an acronym for "Advanced Step in Innovative Mobility", and it is a humanoid robot created by Honda in 2000. It comes from the evolution of other Honda robots, the E series (E0 has been developed in 1986) and the P series robots (ASIMO predecessor is called P4). ASIMO last model has been released in 2011.

It is able to recognize human intention by speech and gesture recognition. It can respond to questions in different languages. It is even able to perform facial recognition. Its most important feature is the ability while walking and running, simulating human movements.

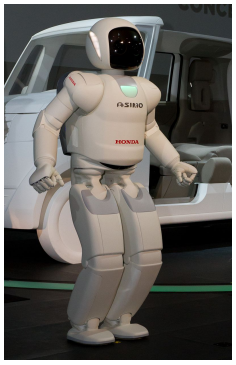


Figure 1.4: ASIMO: the last model

1.4 AIBO

Is it possible for people to like robots without a humanoid body? Looking at Sony experience, yes, we can! AIBO (see [9]), that is an acronym for "Artificial Intelligence Robot" and means "partner" in Japanese, is a series of robotic pets designed and manufactured by Sony. The first consumer model was introduced on 11 May 1999, and Sony is still developing (and selling with success) new AIBO models nowadays too.

AIBO acts like a real dog and can "inspire friendship" in the user by showing its feelings as real pets do. For that purpose, it is able to change its expression and posture. Because of its pretty puppy appearance, it is very useful for pet therapy too.

Recent AIBO models are connected to a cloud, where they store data everyday. They use Machine Learning mechanisms to develop their own "personality" and are able to explore the surroundings and find the shortest path to reach something. They are provided with a camera hidden behind their nose, and they can recognize people. They can perform speech-recognition too, so their behaviour is influenced by what they listen and see, together with memory.

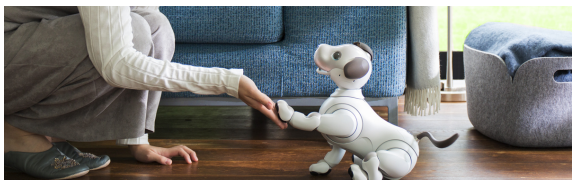


Figure 1.5: AIBO while playing with the user

[8] https://asimo.honda.com/
[9] https://us.aibo.com/

1.5 PaPeRo

PaPeRo (see [10]), which stands for "Partner-type-Personal-Robot", is a personal robot developed by Japanese firm NEC Corporation.

It is a little cute humanoid robot, that seems a puppet. There are different PaPeRo models. The robot's development began in 1997 with the first prototype, called R100.

It can perform facial recognition, using its twin cameras, and can show feeling by moving its neck and lighting up the leds in its body. It can perform autonomous actions. It can move (it can even dance) and perform speech recognition.

It can be used for welcoming people and give information in companies, museums and shops, or at home to keep company. In Japan it is particularly appreciated by children, elderly people and single living alone. It is even used to attend customers and to monitor patients. It can connect to the Internet, and it can be connected to a cloud too to improve its work.

Figure 1.6: PaPeRo welcoming clients

1.6 Nao

Nao (see [11]) is another cute humanoid robot. It is stylized, similarly to PaPeRo, but its body can perform difficult movements while walking and dancing. Its develop has started in 2008. The last model is Nao6.

It is provided with two cameras and can recognize objects and persons and its main goal is the interaction with people. It can simulate feeling with its voice, leds and posture. It is appreciated for research because it is fully programmable. We will talk more about Nao later on this paper.

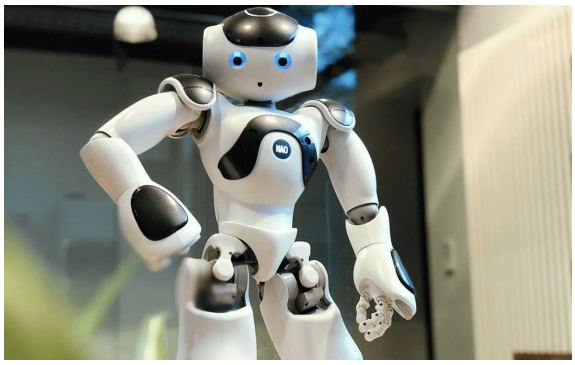


Figure 1.7: Nao6 posing for a photo

A cheap alternative is represented by Alpha2 robot (see [12]).

[10] https://www.necplatforms.co.jp/solution/papero_i/
[11] https://www.softbankrobotics.com/emea/en/nao
[12] https://www.robotshop.com/en/ubtech-robotics.html

1.7 Chatterbots

Chatterbots, or chatbots, are software agents whose main goal is to have a brilliant conversation with people. I used the term "agent" because they are usually based on Artificial Intelligence modules. A common factor behind chatterbots is Machine Learning, in fact they use some technique to learn more complex and better speech ability by studying previously collected data.

Talking with people can be very difficult for a machine, because topics can be a wide variety, and because we can't easily define humor expressions or common saying. Things became even worst if we have to solve that problem for many languages and different cultures. Furthermore, as I previously said, vocal inflections, voice tone, speed in speaking can make the difference.

We are used to think about chatterbots as a software application or a module of a software application, but actually they are integrated on robots that have to talk with people. In the most recent robots models, these speech modules can be boosted up by cloud services.

The fist chatterbot has been developed in 1966 by Joseph Weizenbaum and it is called ELIZA (see [6]). ELIZA simulates a Rogerian psychotherapist by filtering user inputs and responding back with the proper predefined sentence.

Some years later, doctor ELIZA gained its first chatterbot patient, PARRY (see [13]), developed by Kenneth Mark Colby in 1972. PARRY is a simulation of a person with paranoid schizophrenia.

For a more modern approach we had to wait until A.L.I.C.E. (see [7]), an acronym for "Artificial Linguistic Internet Computer Entity", also known as Alicebot. Alicebot was developed by Richard S. Wallace during 2000's and is inspired by ELIZA. It is based on an XML Extention called AIML (see [8]), that is for "Artificial Intelligence Markup Language", which is used to define the heuristic conversation rules. AIML is open source and is available for developers (see [14]).

Jabberwaky (see [15]), developed by Rollo Carpenter in 1997, is the first approach of an entartainer chatterbot.

Nowadays chatterbots are very common and we use them every day, but we call them "assistants". Inputs are no more in text form only, but they can have audio inputs too: they perform audio to text conversion and then they operate on the obtained text with speech recognition operations.

Some examples are:

- Alexa (see [16])
- Google Assistant (see [17])
- Siri (see [18])
- Cortana (see [19])

They are often available for developers as a service to be integrated inside projects.

[13] https://phrasee.co/parry-the-a-i-chatterbot-from-1972/
[14] https://code.google.com/archive/p/aiml-en-us-foundation-alice/downloads
[15] http://www.jabberwacky.com/
[16] https://developer.amazon.com/it-IT/alexa
[17] https://assistant.google.com/
[18] https://www.apple.com/it/siri/
[19] https://www.microsoft.com/it-it/windows/cortana

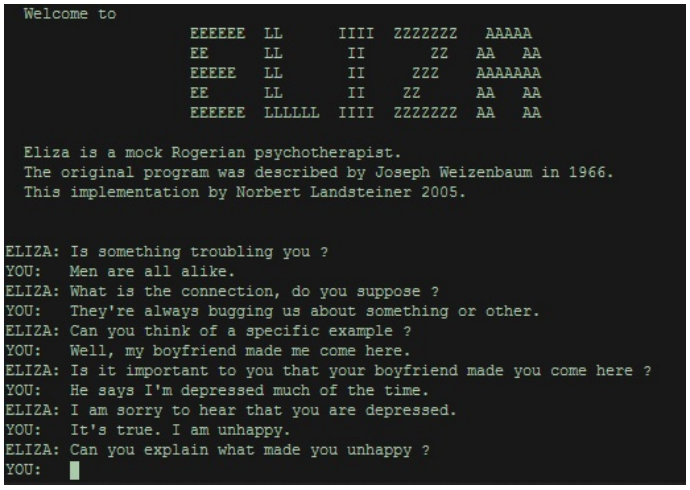


Figure 1.8: A conversation with ELIZA

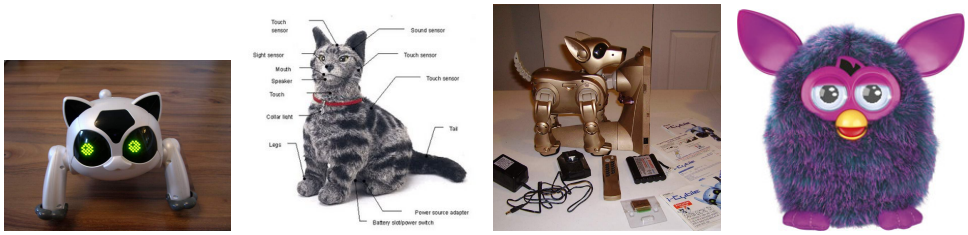
1.8 Robots for children and elderly persons

Then we need to talk about the most exigent clients, children. Many robots are used by children, not for playing only, but as baby-sitters and teachers too.

The first attempts of robots for children are pets. BN-1 (see [20]) is a cat robot toy developed by Banday. It is able to explore the room, take commands and show facial expressions. Its animated LED eyes lights up hundreds of ways in different colors and shapes to express emotions, helped by making kitty tonal sounds too. Touch sensors on paws allowed it to perceive caresses.

Other similar robots are:

- Meow-Chi (see [21])
- Poo-Chi (see [22])
- Necoro (see [23])
- I-Cybie (see [24])
- Furby (see [25]) is one of the more famous toy robot and it has been recently released in a new version



(a) BN-1 (b) Necoro (c) I-Cybie (d) Furby (2012 model)

Figure 1.9: Robot toys

Elderly persons have even more complex necessities, they need entertainment and assistance too. Furthermore, they sometimes are more biased about robots and technology in general.

[20] http://www.theoldrobots.com/bn-1.html
[21] http://www.theoldrobots.com/images64/Meow-Chi.pdf
[22] http://www.robotsandcomputers.com/robots/manuals/Poo-Chi.pdf
[23] https://robotnews.wordpress.com/2007/04/01/necoro-the-lovely-cat/
[24] http://www.theoldrobots.com/icybie.html
[25] https://furby.hasbro.com/en-us

PARO (see [26]) is an advanced interactive robot developed by AIST, a leading Japanese industrial automation pioneer. It allows the documented benefits of animal therapy to be administered to patients in environments such as hospitals and extended care facilities where live animals present treatment or logistical difficulties. PARO has many goals:

- To reduce patient stress and their caregivers
- To stimulates interaction between patients and caregivers
- To improve thier relaxation and motivation
- To improve the socialiazation of patients with each other and with caregivers

It is the World's Most Therapeutic Robot certified by Guinness World Records.

Pepper (see [27]) is a humanoid robot often used as a nurse. It is similar to Nao, but it is taller and it is not provided with robotic legs, but it moves on wheels. Pepper is the world's first social humanoid robot able to recognize faces and basic human emotions. Pepper was optimized for human interaction and is able to engage with people through conversation and his touch screen.

Dinsow (see [28]) is available in different version, and each one can be used differently for patients care. The more recent (and complex) model is Dinsow6.

(a) Woman petting PARO

(b) Pepper

(c) Dinsow allowing a patient to talk with her doctor

Figure 1.10: Robot nurses

[26] http://www.parorobots.com/
[27] https://www.softbankrobotics.com/emea/en/pepper
[28] https://www.dinsow.com/about.html

1.9 Ethics in Robotics

Thinking about Ethics in Robotics, we can't help but remember Isaac Asimov rules. These rules were introduced in one of his short stories, "Runaround", in 1942:

- A robot may not injure a human being or, through inaction, allow a human being to come to harm
- A robot must obey the orders given it by human beings except where such orders would conflict with the First Law
- A robot must protect its own existence as long as such protection does not conflict with the First or Second Laws

In other words, robots should preserve users and people in general.

The more sensitive a person is, the more protective the robot has to be.

A memorable event is My Friend Cayla case. My Friend Cayla (see [29]) is a toy, in particular a doll, which uses speech recognition to have a conversation with children. The doll uses the Internet to get the correct response to what the child said by performin an online research.

The doll is banned in Germany as an illegal surveillance device. A vulnerability in Cayla's software was first revealed in January 2015, a hack allowing strangers to speak directly to children via the My Friend Cayla doll has been shown to be possible (see [30]).

Figure 1.11: Children talking with My Friend Cayla

[29] https://www.genesis-toys.com/my-friend-cayla

[30] https://www.bbc.com/news/world-europe-39002142

Chapter 2

Hybrid Approach

In this chapter I will describe hybrid approach and distributed approach for programming robots, providing its features and converging in Distributed hybrid Applications. I will describe my robotic projects, that can be plugged together or in pre-existent systems to easily solve common robotics problems. They are libraries and frameworks to build distributed hybrid systems where logic programming modules, MASs, Neural Networks and procedural/Object Oriented modules can eventually be plugged and work together easily, both in simulation and real world environments. There are a library for managing drones and an hardware/software project for robots that have to work in areas where Internet connection is not available. Their code is available on GitHub (links will be provided).

Thinking about the software for a fully working robot, we could imagine a big complex application, furthermore emotion recognition and emulation make things much more difficult. A robot usually has a big amount of tasks, even if its main goal could be only one. For example, a robot that is in charge of people tracking has to manage its camera and analyze the incoming images, but it maybe has to move along the room at the same time. So, together with its special tasks, the robot has to perform other involved actions: moving without falling and standing up in case of unwanted falling, obstacle detection and avoidance, manipulating objects without breaking them, and so on.

It is obvious that the developer can't easily work on the whole software together. Probably it would be impossible to think about the whole problem at once too. According to an important Software Engineering good practice, each robot task has to be observed individually. If the task is still very complex, it has to be divided in sub-tasks, and so on, in order to have affordable problems. That concept is known as "divide et impera" (divide and rule). After tasks fragmentation, developers have to handle the obtained different code parts, called module. Modularity is appreciated in software development and it is a design metric.

Each module can be very different, so developers actually don't have to use the same technologies to make them. They can use different programming languages, frameworks and even programming concepts.

They can write code by:

- specifying the process step by step, following procedural programming way
- describing the start state, the wanted state and the rules that can determine the change between adjacent states, following logic programming way
- creating the module and training it, using machine learning way (for example, using neural networks)

No one of these approaches is better than the others, but their usefulness can vary according to the goal that the module has to accomplish. So it is very reasonable to properly mix different programming techniques, keeping in mind they have to be able to work with each other. For this purpose, we need, in particular, to use a proper format for data, in order to create a valid and efficient common communication standard for modules.

In this section I will give some examples of possible solutions for hybrid modular programming by showing some of the projects I experimented on.

Another objective we have to keep in mind is making software that can be easily processed by robots. Thinking about robot hardware, we can define a robot as a set of sensors, processors and actuators, but we cannot go deeply in describing it because robots are very different from each other. Each model has its own hardware according to tasks, environment, users, manufacturers, price. For this reason too, a developer has always to face with specifics and hardware limitations. Another important feature in robot software portability, or rather it has to correctly run on as many robots as possible, so it has to be general.

Each hardware has its own characteristics and its own limits as well and developers have to write optimized code in order to avoid resource wasting: memory, time, processing capacity and even electrical power are not infinite. These resources are needed by all robots main general tasks: sensing, planning and acting, each one maybe composed of different sub-tasks. Keeping well in mind "divide et impera" philosophy, we can apply it to this context too by performing a division of these tasks and sub-tasks. How? Creating a distributed system. A distributed system is a set of processes that can run on different machines and exchange data via a common broker.

A distributed hybrid system is a hybrid system that is distributed over different machines. It is more performing in relation with a not distributed hybrid system and with a standard distributed system, indeed it uses a different technology on different machines according to their task and to the specific hardware and it can run in parallel on different machines in order to distribute the workload. Obviously, as in everything, there is also a negative aspect, that is complexity for developer in processes coordinating and communication.

In the remainder of this chapter I will survey a repertoire of diverse technological solutions, along with personal experiences on specific modules based on them. The aim of the discussion is not to present an exhaustive list of solutions (well beyond the scope of this thesis) but rather to single out a set of distributable modules, compatible with each other and easily pluggable even in pre-existing environments. All these modules are available on GitHub and can be used to build hybrid systems: you can see them like different instruments you can use for easily developing a distributed hybrid system without warring about technological compatibility, data format, communication channel, providing input data mode and result retrieving mode. In particular, I will give some insights to Multi Agent Systems, Neural Networks, Robot and Drones, and Robot in rural areas, concluding the chapter discussing some proposals of mine for favoring the utilization of a hybrid approach in simulation environments.

2.1 MAS

The term MAS stands for "Multi Agents System" and indicates an application based on multiple processes, called Agents.

These agents are daemon processes, in fact the are always active, running in the background. They are proactive, because they properly react to external and internal events. External events are given by the environment the agent lives in. Internal events instead depend on the agent behavior and can be triggered by the occurrence of a particular condition of its current state.

The so called intelligent agents are agents that use some Artificial Intelligence technique, for example they can be based on logic programming techniques.

They can communicate in order to exchange knowledge and collaborate with each other to easily reach their goals. Main goals are defined by the program (or logic rules), but there are other sub-goals the agent has to reach in order to satisfy main goals requirements. An easy example of goals and sub-goals relation could be the following: if a robot has the main goal to reach an object and take it, sub-goals are: locate the object, planning a path to reach it, performing each step while moving, and so on.

In a real environments, this sequence of action is not so simple because it actually is a cycle. In fact, external world is not static like in the previous example, it is dynamic and can continuously change its state. For example, the robot should sometimes relocate the target object, because maybe it has been moved somewhere else, and consequently should make a new plan for the new position to reach and maybe change its direction. Sensing, planning and acting are periodically performed. The smaller is the period, the more efficient is the robot, but the bigger is the period, the more resource consuming is the robot, so developer has to find a good balance considering the context of use.

Sometimes working with MAS hybrid systems is difficult because each MAS technology uses a different communication channel and data format, so data exchange can be challenging for

everything that is external to the MAS: agents easily talk each other, but they are unreachable by other modules or programs, so they are not easily integrable, especially in pre-existing systems.

For my previous thesis I worked on ServerDALI, a MAS - Procedural Programming hybrid approach. This first system is based on a DALI MAS (see [1]), integrated in a web server running in a Docker Container(see [2]). DALI is a framework based on Prolog, in particular SICStus Prolog (see [3]), a logic programming language. I used Docker Container technology to make the system as portable as possible.

ServerDALI is composed of:

- A simple DALI MAS (see [4])
- A Prolog parser, called ServerPROLOG (see [5])
- A PHP web server (see [6])

All these modules are included in a Docker container (see [7]).

Then, I had the opportunity to participate to Intelligent Systems And Robotics Laboratory course and I finally improved it, developing more efficient modules for Python 3 applications and making it able to be distributed over many machines. This system is based on different MASs, programmed with different languages, and Object Oriented modules, all connected together via a broker based on Redis (see [8]).

This new version of the system, called KOINE DALI, that is an extension of DALI framework I developed during research, is described in the paper, available at the following link: [9]

An easy boilerplate is available on GitHub (see [9]).

It is based on some Python3 and Prolog libraries I developed (see [10]).

Thanks to Redis integration, this version of DALI is modular, distributable and scalable and you can easily add it to a pre-exixtent application, independently on the technologies used by other modules.

MASA agent is a special agent included in each MAS. It can receive Redis brocker messages directly and is in charge of message delivery inside the MAS: it assigns each incoming message to the proper agent.

[1] https://github.com/AAAI-DISIM-UnivAQ/DALI
[2] https://www.docker.com/
[3] https://sicstus.sics.se/
[4] https://github.com/agnsal/ServerDALImas
[5] https://github.com/agnsal/ServerPROLOG
[6] https://github.com/agnsal/ServerDALI
[7] https://github.com/agnsal/docker-ServerDALI
[8] https://redis.io/
[9] https://github.com/agnsal/KOINE-DALI
[10] https://github.com/agnsal/Redis2LINDA-stringESE

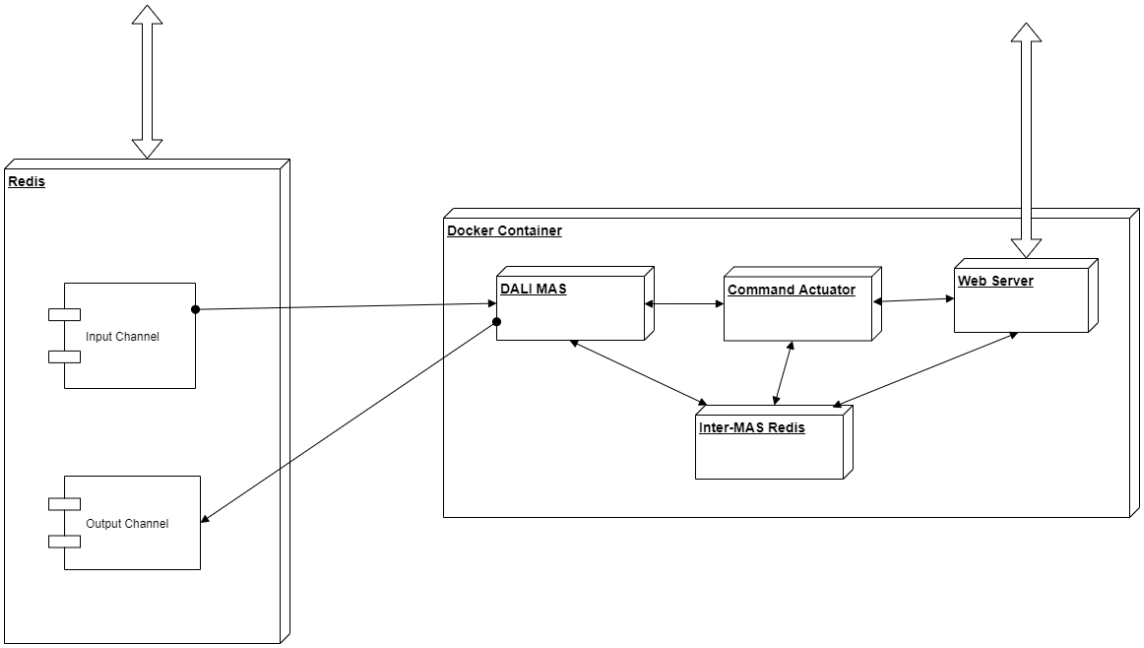


Figure 2.1: Koinè DALI General Deployment Diagram

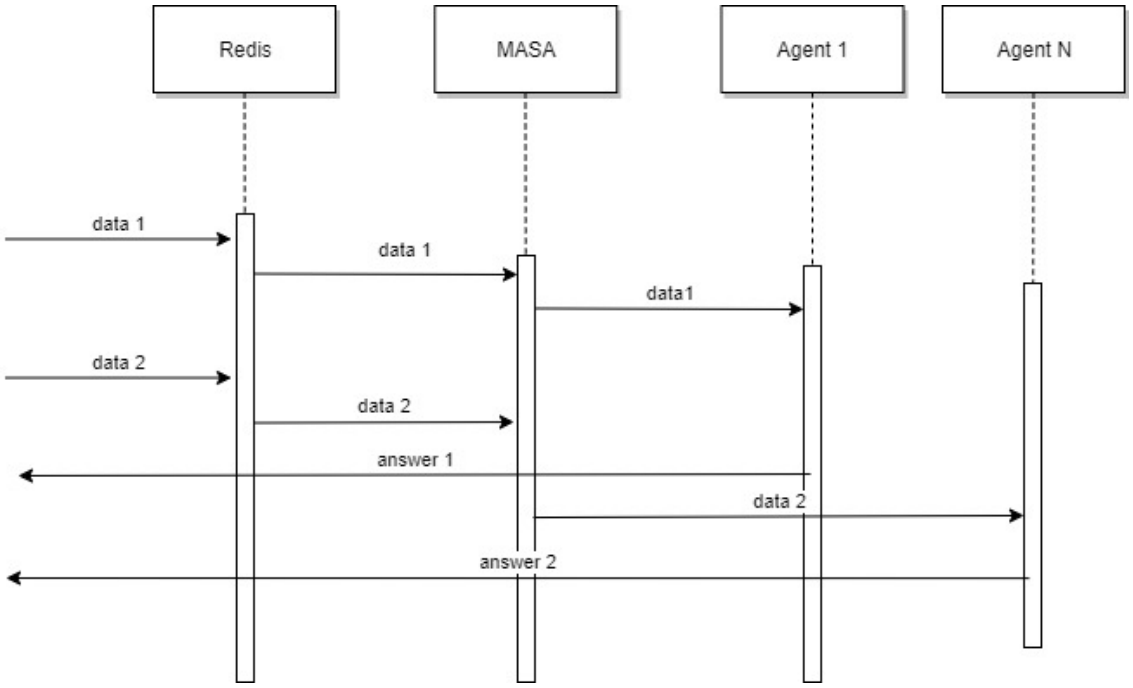


Figure 2.2: Koinè DALI Sequence Diagram

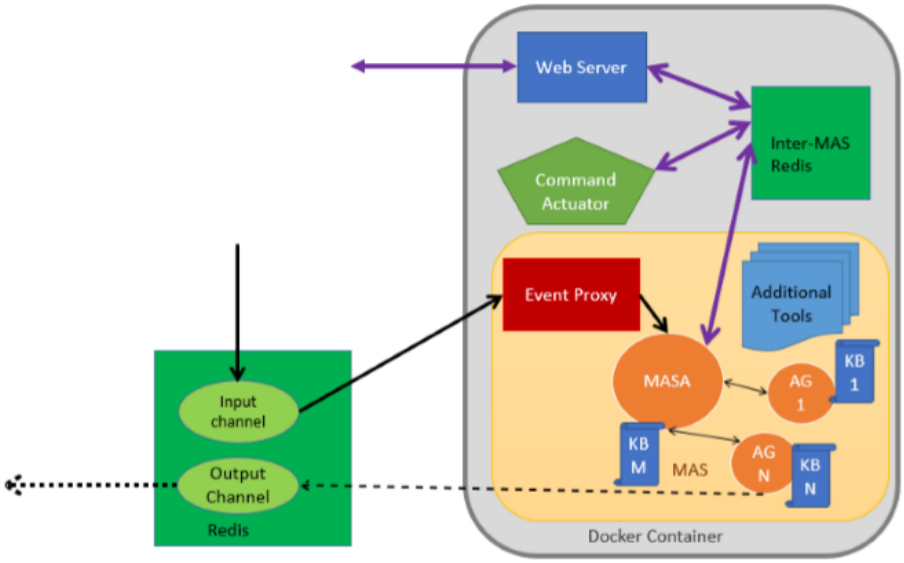


Figure 2.3: Inside a Koinè DALI MAS

2.2 Neural Network

A Neural Network, or Neural Net, is a set of interconnected neurons and is used for solving Artificial Intelligence problems.

The connections (synapses) of the these neuron are modeled as weights. A positive weight reflects an excitatory connection, while negative values mean inhibitory connections. All inputs are modified by a weight and summed. This activity is referred as a linear combination. The amplitude of the output is controlled by an activation function.

Neural Networks are often used for making predictions and classifications over big data. Their efficacy may depend on their structure, that is the number of neurons, their distribution and the number of synapses, and on dataset (in particular on dataset quality and dimension), solved problems the nets train on before working.

To allow working with both MASs and Neural Nets, during Machine Learning course I worked on NeuralMAS project (see [11]).

This system includes:

- A Keras Neural Net
- A Koinè DALI MAS
- A Redis Broker

I chose Keras (see [12]) because is a powerful and versatile Python 3 library, providing an high-level API for Neural Networks. It can run on top Theano, CNTK and Tensorflow. For my project I used Tensorflow (see [13]), a platform for machine learning.

The advantage of using Keras is the ability to change the underlying technology without impacting the code. That is useful when underlying technology updates as well, in fact Keras manages all the problems related to changes.

As I previously said, all my modules can be used together in the same system.

[11] `https://github.com/agnsal/NeuralMAS`
[12] `https://keras.io/`
[13] `https://www.tensorflow.org/`

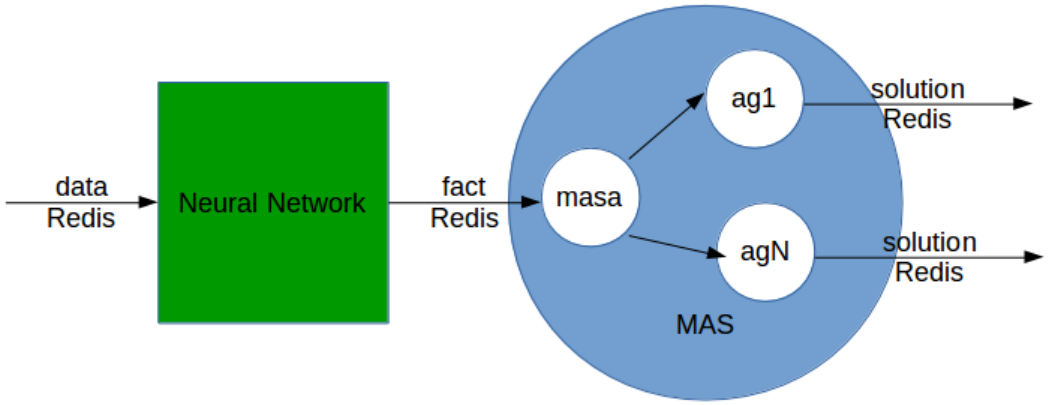


Figure 2.4: Inside a Neural Redis System

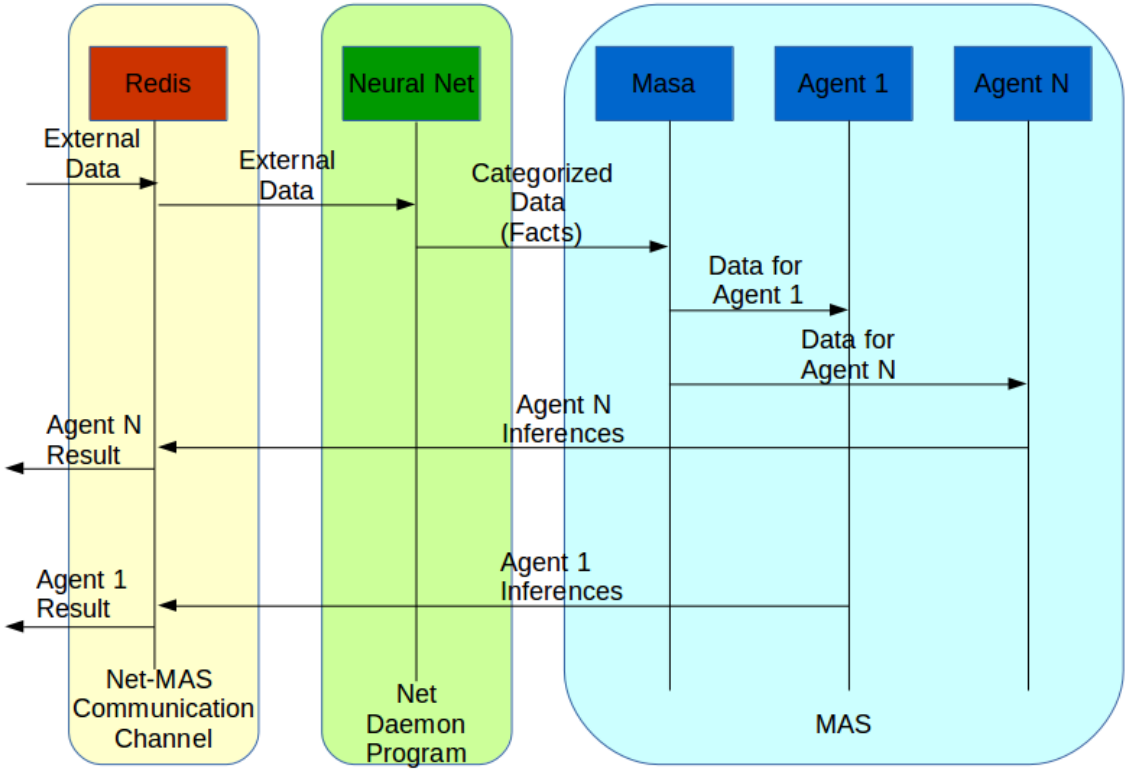


Figure 2.5: Neural Redis Sequence Diagram

2.3 Robots and Drones

Drones are a particular kind of robots that usually are not provided with a reasoning module: they are guided by user.

If we think about drones we may not expect that drones can be part of a hybrid system, but what if drones are seen as robots extensions or tools? Drones could be guided by other systems or at least could send data for these systems, that are maybe hosted by robots or that are hosted by a server called by robots.

For example, during Modelling and Control of Communication Networks course, I had to work with little Crazyflie 2 drones (see [14]), so I developed CrazyRedis, a Python 3 library to easily manage Crazyflie 2 API with callbacks and to receive log data via Redis.

The project is available on GitHub (see [15]).

It is fully compatible with my other projects.

2.4 Robots in Rural Areas

Sometimes robots have to work in areas where the Internet is not available, for example in rural areas or inside buildings. How should a robot act when such a situation occurs?

A developer has always to keep in mind work context and robot tasks. If the context is variable, maybe a robot can fully work when some condition is met, for example when an Internet connection is available, but the Robot basic functions must be always guaranteed. Emergency situations are included in context changes too.

LoRaWAN networks are good alternative to the Internet when data to deliver are not very big. LoRa, that stand for Long Range, is a wireless technology to transmit small data packages (0.3 to 5.5kb/s) over a long distance with low power consumption. LoRa is used for LPWAN, that is Low Power Wide Area Network. LoRa technology consists of radio transmission, based on spread spectrum modulation.

LoRaWAN is the network technology built upon LoRa physical layer, in particular it is the media access control protocol for the management of the communication between the LPWAN gateway and the end-devices:

- It is managed by LoRa Alliance
- It defines the network topology, the communication protocol, communication frequencies, data rate and power management aspects
- The devices are asynchronous
- Data coming from an end-device may be received by many gateways, that are in charge of sending that data to a centralized server, that has to filter duplicate packets, perform security checks and manage the network
- Finally, data are delivered to application servers

[14] `https://www.bitcraze.io/crazyflie-2/`

[15] `https://github.com/agnsal/CrazyRedis`

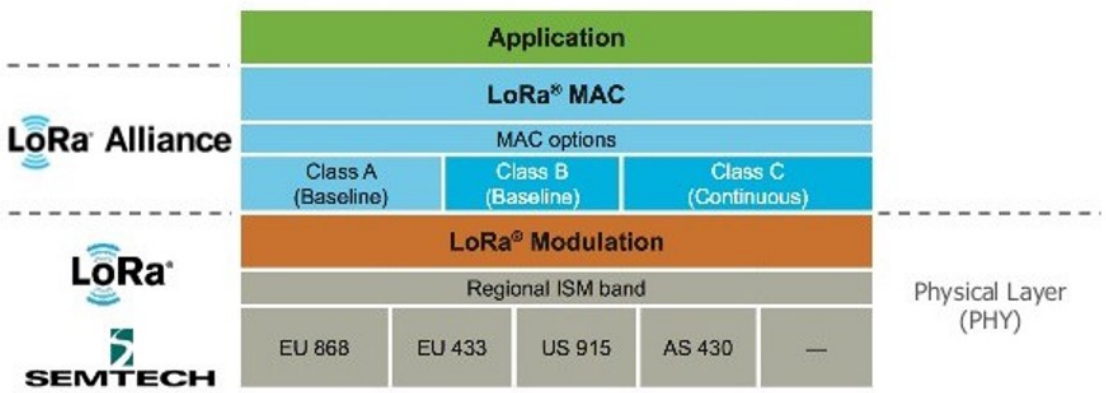


Figure 2.6: LoRa and LoRaWAN

LoRaWAN networks have a star topology:

- End-Nodes are connected to one or more gateways (or concentrators)
- Each gateway (star-center) is connected to the Internet and can interact with servers

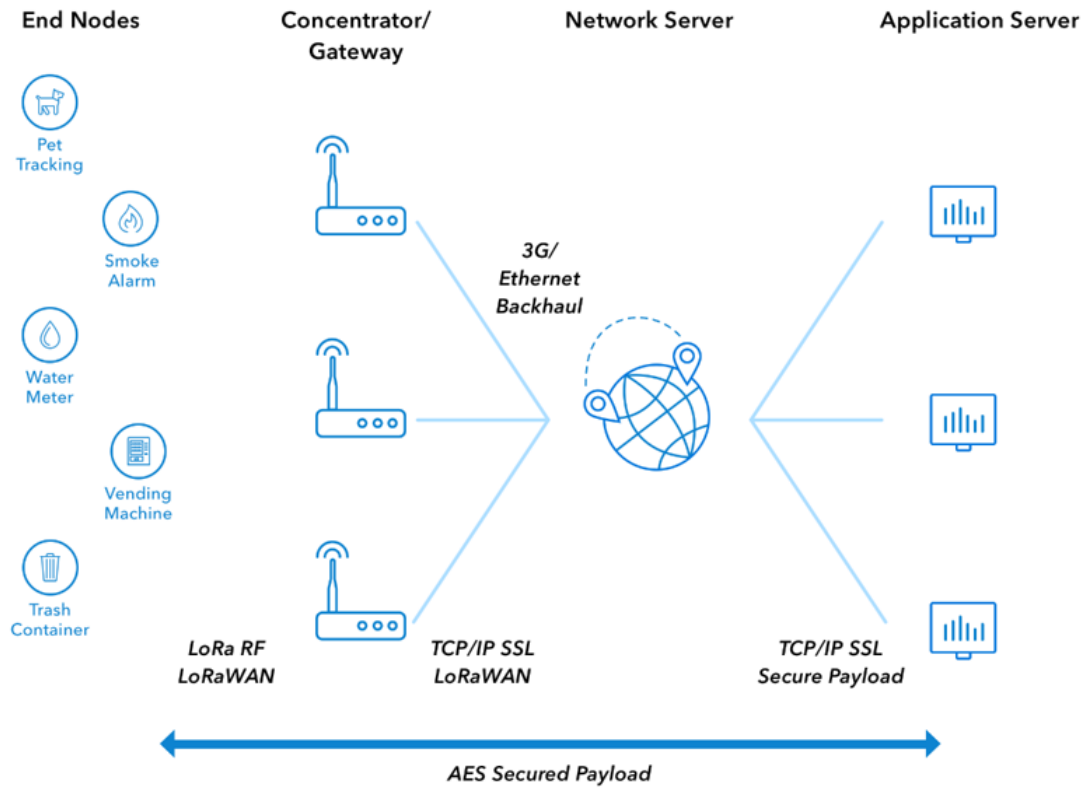


Figure 2.7: LoRaWAN

I developed a LoRaWAN hardware/software project, Lora Robot Rescue, during Embedded Systems course. This project is available on GitHub (see [16]).

It is based on Arduino MKR 1300, so I wrote its software module in C/C++. Data coming from LR2 can be delivered to Redis by a proper application, so this module is fully compatible with my other projects.

[16] `https://github.com/agnsal/LoraRobotRescue`

Lora Robot Rescue, or LR2, has the purpose of solving most common and annoying outdoor robot problems. In fact, working with outdoor robots, especially with rovers and drones, users are always facing with:

- Limited energy autonomy
- Unpredictable breakdowns
- Troubles with communication in rural areas

Furthermore, if one or more of the previous conditions occurs, how can the user find his (maybe very expensive) no more traceable robot?
LR2 is a module that can be installed on a generic Robot to:

- Periodically send the GPS location, the charge value of the battery and robot temperature via a LoRaWAN connection
- When, for example, power is under and/or temperature is over a certain threshold, send an alert (containing these parameters) and signal the robot to enter in powersafe mode
- Make the robot more evident, by lighting and emitting sounds, when robot owner is looking for it

Another advantage:

- LR2 could be a nice antitheft for robots

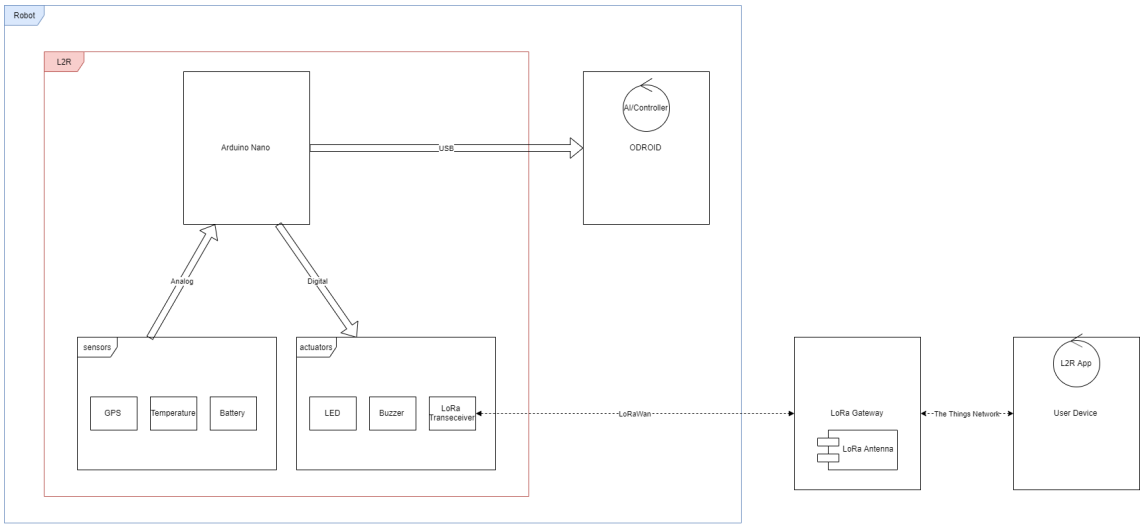


Figure 2.8: LR2 structure

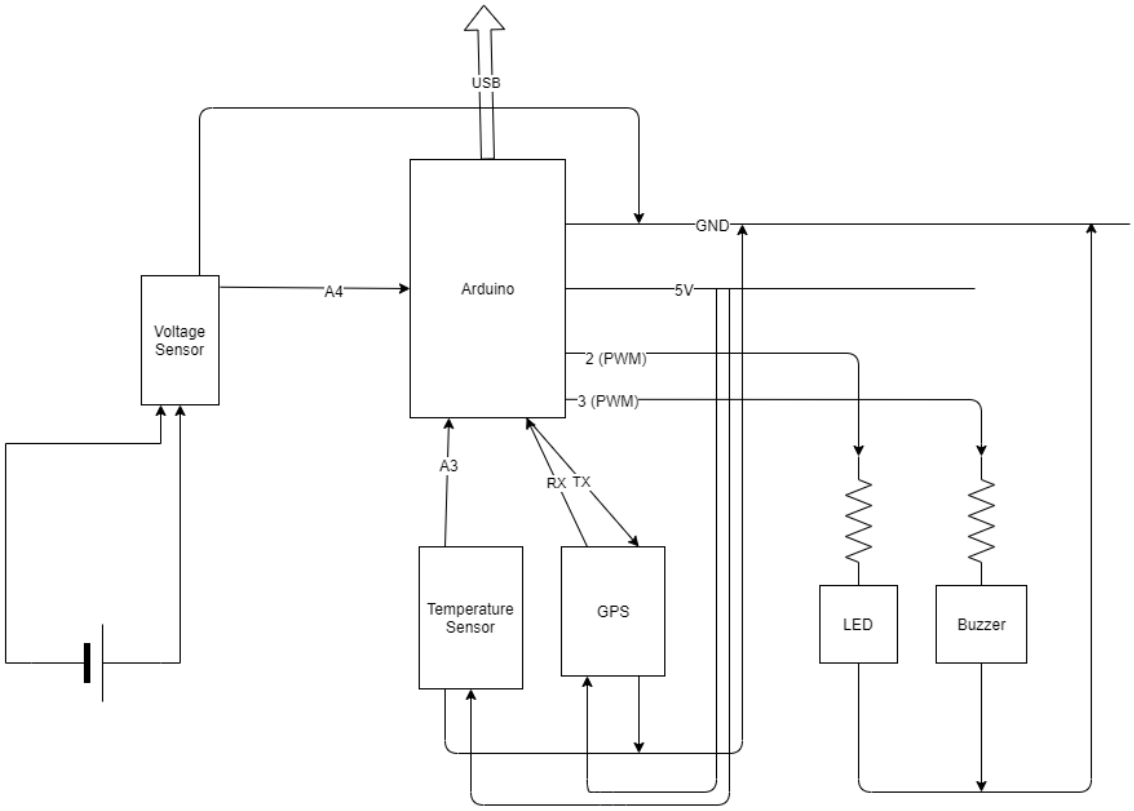


Figure 2.9: LR2 Hardware scheme

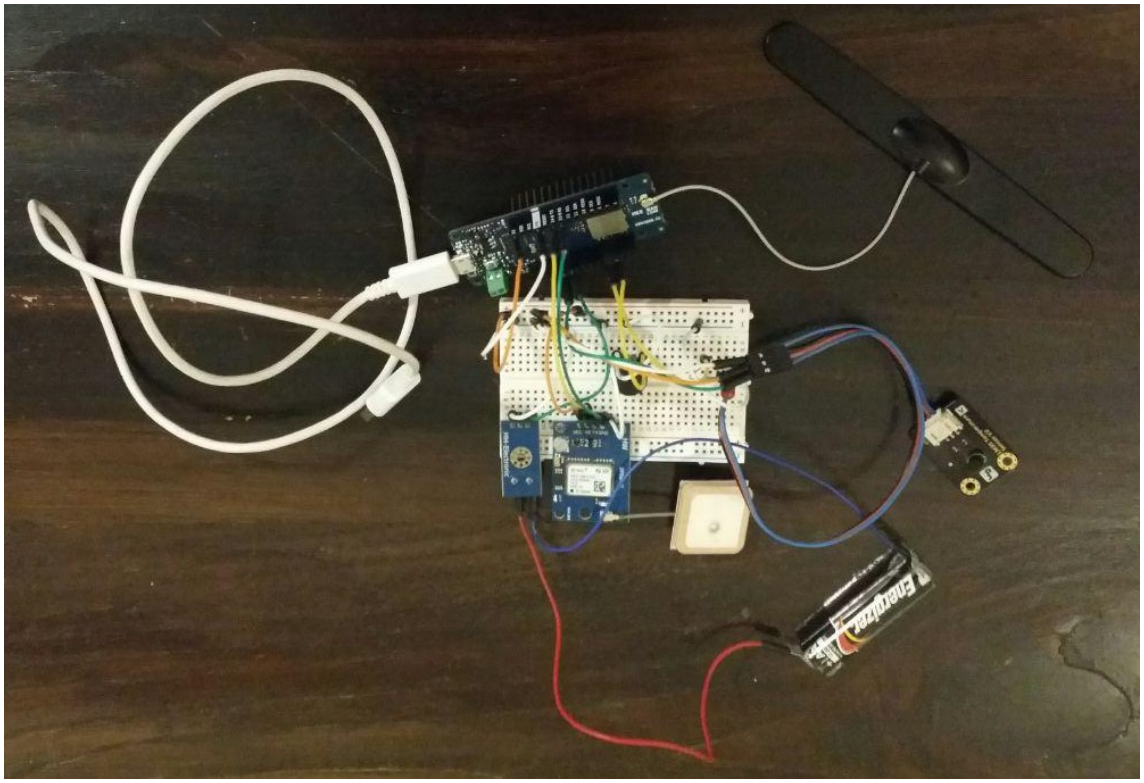


Figure 2.10: LR2 Hardware

2.5 Hybrid Approach Simulations

The very first step in robot software development is prototyping, but doing it on real robots is very expensive, because robots can be damaged in such a process.

Simulation environments are born to avoid this problem, but its difficult to make an hybrid application work inside them.

I developed Prothonics-vrep and its successor, Prothonics, to allow users to create agents that can perform reasoning using SWI-Prolog in V-REP/CoppeliaSim simulations. V-REP and Coppeliasim are simulators from Coppelia Robotics (see [17]) and are widely used.

Protonics-vrep and Protonics are both available on GitHub (see [18] and [19]).

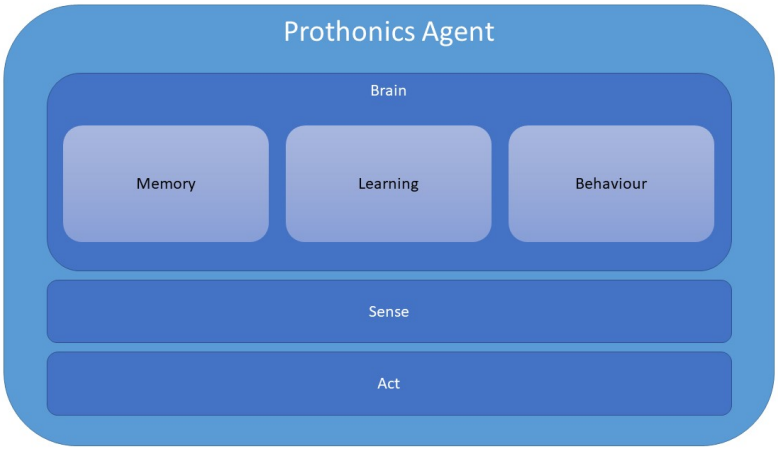


Figure 2.11: Prothonics

Another well known simulation environment is Gazebo, that is embedded in ROS, Robot Operating System. It is very useful, because it allows to develop code for simulation and real robot at once (usually code for simulation is different from the one for real robots), but it has a lot of compatibility problems:

- Each newer versions is not retro-compatible
- Each version requires a different Linux Operative System version
- Developer has to run the same version of ROS (and so the same OS version) both on his computer and on Robot
- Each robot model requires its version of ROS (if it supports ROS), so developers should have many different OSs installed or virtualized on his computer, and maybe some of them are obsolete

For my Intelligent Systems And Robotics Laboratory course project (see [20]) I and my colleagues had to work with ROS, so I developed the following modules:

- Docker-IndigoROSdisPyPl (see [21]), a Docker container with all the needed toold to create Python, Python/SICStus-Prolog, Python/SWI-Prolog and Python/Datalog agents in a ROS environment
- KobukiROSindigo (see [22]), an example of such a system

SICStus Prolog, SWI-Prolog and Datalog are different implementations of Prolog logic programming language.

[17] https://www.coppeliarobotics.com/
[18] https://github.com/agnsal/prothonics-vrep
[19] https://github.com/agnsal/prothonics
[20] https://github.com/C-A-V-ROBOT/DR2
[21] https://github.com/agnsal/docker-IndigoROSdisPyPl
[22] https://github.com/agnsal/kobukiROSindigo

Figure 2.12: Docker-IndigoROSdisPyPl

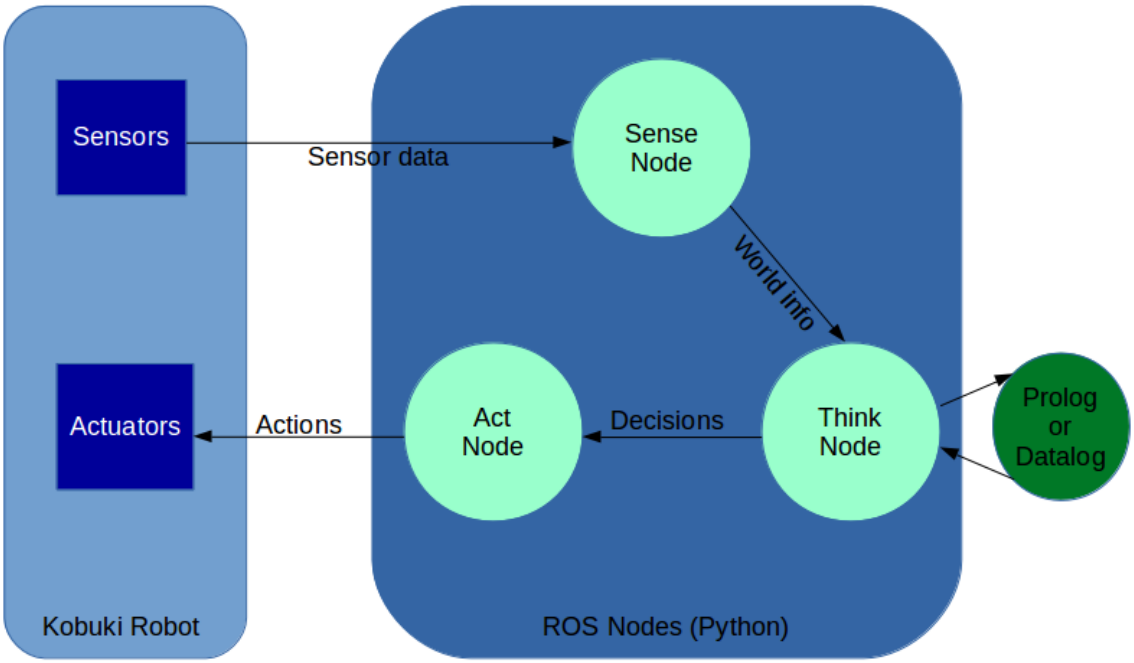


Figure 2.13: KobukiROSindigo Node Diagram

Chapter 3

BlocksBot: Simulated and Real Empathic Robot Project

In this chapter I will present and describe BlocksBot, my Empathic Robot project. First of all, I will present technologies I used to realize it, that are the message broker, that is Redis, the image and audio processing tools, that are Face++ and Vokaturi, and the real robot I used, Nao. Then I will present my project module by module in details. Code is available on GitHub at the given link.

BlocksBot project aims to provide robots a technology to perform emotion detection and to react to detected emotion. In this first version, BlocksBot detects emotion by analyzing facial expression, voice and body posture.

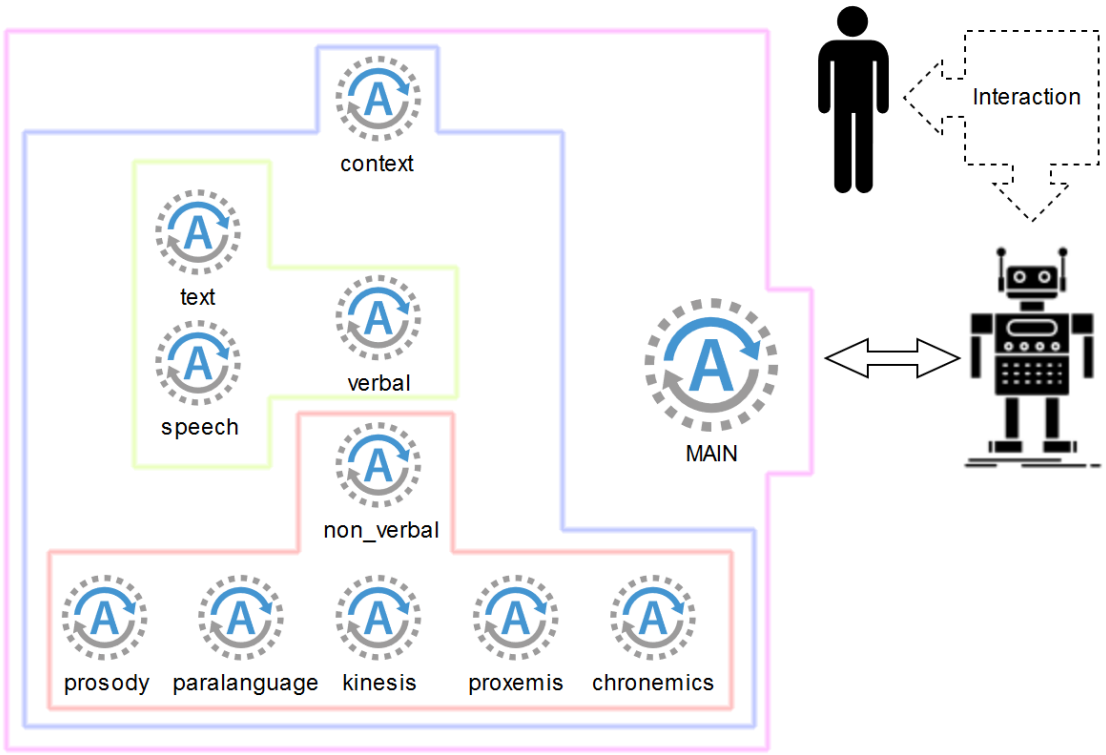


Figure 3.1: Multi-agent system organization to implement an empathic human-robot interaction. The agents communicate according to the hierarchy of the 8-dimensions communication model: verbal (test, speech), non_verbal (prosody, paralanguage, kinesis, proxemis, chronemics), and context. BlocksBot includes 2 of them: prosody and kinesis, that are both non_verbal

Nao reactions can be easily configured, modified or expanded. In this first experiment, Nao

reactions consist in saying which is the detected emotion while emulating happiness if the emotion is positive, doubt if it is neutral and sadness if it is negative. This emulation is performed thanks to Nao Animated Speech feature, that provides proper animations. For simulation robot, instead, reaction consist of writing the detected emotion in a simulation window while performing the task of looking at the user, that moves in front of the webcam, as during a video call.

NAO - List of tags available by default

Tag	Matching animations
affirmative	Yes_1; Yes_2; Yes_3
alright	Yes_1; Yes_2; Yes_3
beg	Please_1
beseech	Please_1
body language	BodyTalk_1; BodyTalk_1; BodyTalk_10; BodyTalk_10; BodyTalk_11; BodyTalk_11; BodyTalk_12; BodyTalk_12; BodyTalk_13; BodyTalk_14; BodyTalk_15; BodyTalk_16; BodyTalk_17; BodyTalk_18; BodyTalk_19; BodyTalk_2; BodyTalk_2; BodyTalk_20; BodyTalk_21; BodyTalk_22; BodyTalk_3; BodyTalk_3; BodyTalk_4; BodyTalk_4; BodyTalk_5; BodyTalk_5; BodyTalk_6; BodyTalk_6; BodyTalk_7; BodyTalk_7; BodyTalk_8; BodyTalk_8; BodyTalk_9; BodyTalk_9; Explain_1; Explain_10; Explain_11; Explain_2; Explain_3; Explain_4; Explain_5; Explain_6; Explain_7; Explain_8; YouKnowWhat_1; YouKnowWhat_5
bow	BowShort_1
call	Hey_1; Hey_6
clear	Explain_1; Explain_10; Explain_11; Explain_2; Explain_3; Explain_4; Explain_5; Explain_6; Explain_7; Explain_8
enthusiastic	Enthusiastic_4; Enthusiastic_5
entreat	Please_1
explain	Explain_1; Explain_10; Explain_11; Explain_2; Explain_3; Explain_4; Explain_5; Explain_6; Explain_7; Explain_8; YouKnowWhat_1; YouKnowWhat_5
happy	Enthusiastic_4; Enthusiastic_5
hello	Hey_1; Hey_6
hey	Hey_1; Hey_6
hi	Hey_1; Hey_6
I	Me_1; Me_2
implore	Please_1
indicate	You_1; You_4
me	Me_1; Me_2
my	Me_1; Me_2
myself	Me_1; Me_2
negative	No_3; No_8; No_9
no	No_3; No_8; No_9

Figure 3.2: Nao Animated Speech documentation extract

BlocksBot is based on different distributed Python 2/3 modules and on a Redis broker to assure:

- Portability
- Modularity
- General, high-level development
- Great compatibility

BlocksBot provides tools both for simulation and for real robots.

3.1 Redis

In the BlocksBot project I used Redis as a data message broker, like for my previous ones. This choice assures compatibility between all my projects, that are hybrid application modules that can easily be assembled and work together, in local or distributed systems. Redis assures compatibility with other applications as well. Redis is an increasingly used technology and is performing in distributed applications.

Redis (see [1]) is an open source (BSD licensed), in-memory data structure store, used as a database, cache and message broker. It supports data structures such as strings, hashes, lists, sets, sorted sets with range queries, bitmaps, hyperloglogs, geospatial indexes with radius queries and streams.

Currently, Redis has much more features compared to its previous versions. A Python library, called Redis, provides an easy to use API and it properly handles polling related problems, allowing asynchronous processes even during listening tasks.

[1] `https://redis.io/`

Listening tasks consist of waiting for messages on a channel, after submission. Messages can be published on Redis channels for delivery, similarly to other brokers.

My current Redis version is 3.2.100 and I am running it on Windows 10 OS. Redis can run both on Windows and Linux Operating Systems and is fully backward compatible.

Redis can be secured by configuring a password for clients. It can be configured for data encryption via SSL/TLS, that is supported by Redis starting with version 6. Redis clients communicate with the Redis server using a protocol called RESP (REdis Serialization Protocol). While the protocol was designed specifically for Redis, it can be used for other client-server software projects. RESP is a compromise between the following things:

- Simple to implement
- Fast to parse
- Human readable

RESP can serialize different data types like integers, strings, arrays. There is also a specific type for errors. Requests are sent from the client to the Redis server as arrays of strings representing the arguments of the command to execute. Redis replies with a command-specific data type.

Redis relies on TCP at Network Layer. A client connects to a Redis server creating a TCP connection to the port 6379. Port number can be changed in configuration too. While RESP is technically non-TCP specific, in the context of Redis the protocol is only used with TCP connections (or equivalent stream oriented connections like Unix sockets).

Redis accepts commands composed of different arguments. Usually, once a command is received, it is processed and a reply is sent back to the client, so the Redis protocol is a simple request-response protocol, however there are two exceptions:

- Redis supports pipelining, so it is possible for clients to send multiple commands at once, and wait for replies later
- When a Redis client subscribes to a Pub/Sub channel, the protocol changes semantics and becomes a push protocol, that is, the client no longer requires sending commands, because the server will automatically send to the client new messages (for the channels the client is subscribed to) as soon as they are received

While using Redis, there is an important matter to monitor: memory usage. It is both a broker and a NoSQL database at the same time, and it can be very efficient as both, but if memory usage increases it will be slower in message delivery. So, to develop a good system with Redis and using it as a broker too, I would recommend to use it as a temporary memory and to move data to a different database if you need persistence of a big amount of information. Redis memory can be cleaned by assigning a timeout parameter to entered data or by designing a proper input - output mechanism, for example by balancing push - pop calls, by flushing it at regular time intervals or by assign a timestamp to stored data and periodically clean old inputs.

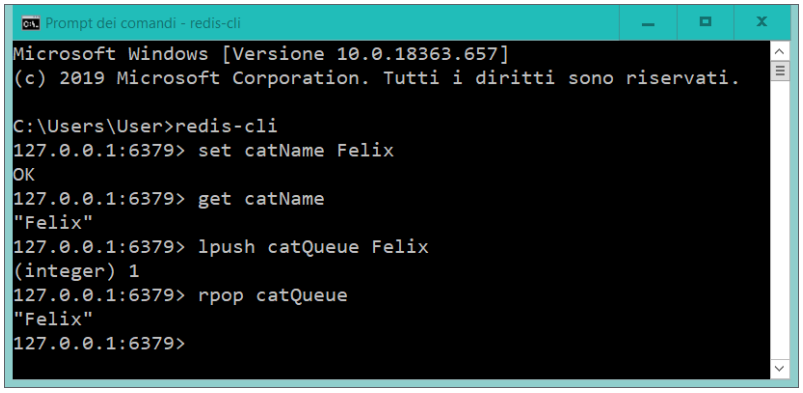


Figure 3.3: Redis Cli usage example

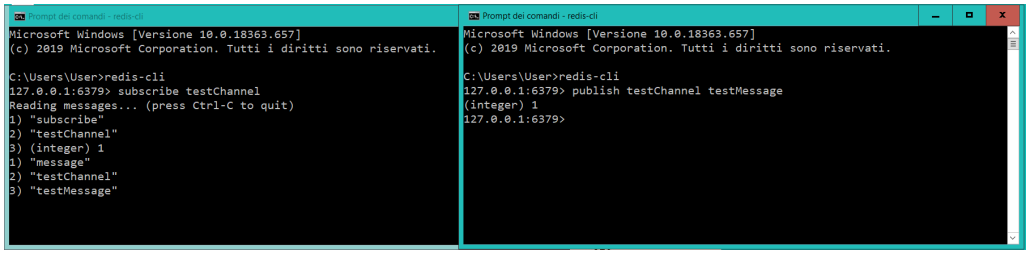


Figure 3.4: Redis Publish - Subscribe example

3.2 AI tools for emotion detection

BlocksBot aim is to detect user current emotion by deducing it from images and audio records. Image and audio processing are complex tasks that are usually managed by Neural Nets. A good Neural Net, as I told before, needs a big amount of data as dataset to perform training, and good data are not so easy to collect. In cases like this, leaning on services that provide pretrained and very efficient Neural Networks is the preferable choice, when applicable.

Face++ (see [2]) is a provider of many services related to image processing. It is based on Neural Networks and it is callable via HTTP requests, but it is available as an SDK for offline use too. Developers have to pay for SDK or online full version, but can try all their online services for free by creating their own account. I have obviously chosen this last option.

I used Face++ Face Detection API to detect faces and get their emotion attributes: a Json containing seven emotions and their percentage probability. Available emotions are:

- Anger
- Disgust
- Fear
- Happiness
- Neutral
- Sadness
- Surprise

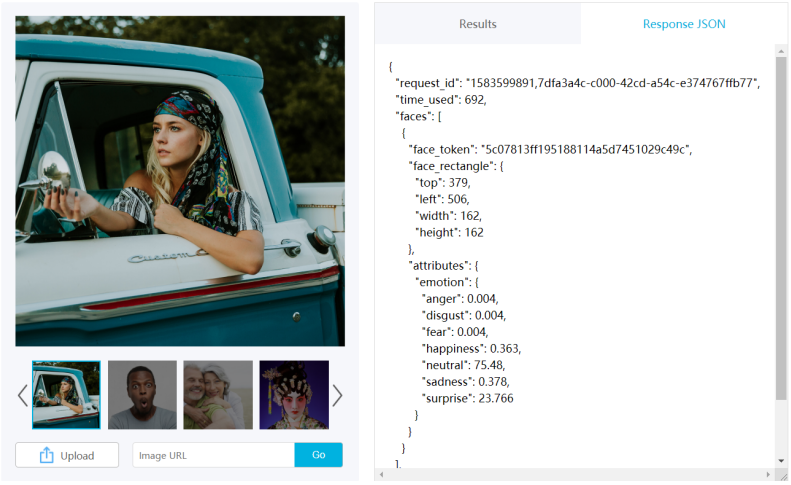


Figure 3.5: Face++ Face Detection API example

[2] https://www.faceplusplus.com/

Another Face++ service I used for BlocksBot is Skeleton Detection. Skeleton Detection detects human bodies and their parts (like head, shoulders, and so on), returning their coordinates. In this case, Face++ doesn't provide emotion probabilities. I get an attitude value by detecting distance between hands: when hands are close each other a person has an introverted attitude, otherwise he has an extroverted attitude.

Detect API is very reliable, unlike Skeleton Detection.

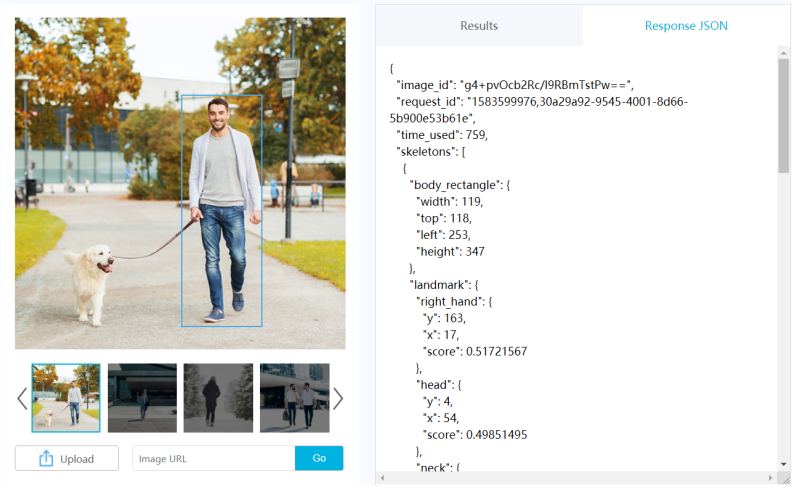


Figure 3.6: Face++ Skeleton Detection API example

For voice analysis, I used Vocaturi service (see [3]). It can analyse speech emotions and it is independent of the language of the speaker. Vokaturi is available as an SDK and can be easily integrated into existing software applications. Full versions, VokaturiPlus and VokaturiPro, can recognize seven emotions and have an accuracy of 76.1 percent. Free opensource version, OpenVokaturi, can recognize five emotions and has an accuracy of 66.5 percent.

Full version emotions are:

- Neutral
- Happiness
- Sadness
- Anger
- Fear
- Disgust
- Boredom

Free version emotions are:

- Neutral
- Happiness
- Sadness
- Anger
- Fear

Like for Face++ services, Vokaturi emotions are accompanied by their probabilities.

[3] `https://vokaturi.com/`

3.3 Nao

For this project, I used Nao 5 (see [4]), a humanoid robot. NAO is the first robot created by SoftBank Robotics. Famous around the world, NAO is a tremendous programming tool and he has especially become a standard in education and research, in fact it is an open and fully programmable platform. NAO is also used as an assistant by companies and healthcare centers to welcome, inform and entertain visitors, so it is very suitable for BlockBot's purpose.

Its body is characterized by 25 degrees of freedom, which enable him to move and adapt to his environment. It can dialogue available in 20 different languages, the one used for the project is configured for Italian language. It has four directional microphones and speakers to interact with user. It has two 2D cameras that allow to recognize shapes, objects and even people.

Last Nao version supports Python 3, but Nao 5 can be programmed in Python 2 only.

Figure 3.7: Professor De Gasperis with laboratory Nao, called Naetto

3.4 BlocksBot agents

BlocksBot is a made of different sets of agents:

- The first group is made of 4 agents: 3 of them have have to analyse images and audio recordings in order to detect emotions and the other agent has to combine their results in order to decide which is the most probable emotion
- The second group is made of 2 agents and has to manage CoppeliaSim simulation, capturing images with the webcam and recording audio
- The last group is made of 3 agents that has to run on real robot: 2 of them have to capture images and record audio, indeed the last one has to perform the proper reaction to current emotion

Finally, there are 2 special agents, Runners, that have to start and stop all the other agents: one is for simulations and the other one is for real Nao.

I am going to describe all these agents in details.

[4] `https://www.softbankrobotics.com/emea/en/nao`

The first group of BlocksBot agents includes four agents that can be used both on real robots or other devices, like computers:

- DecisionMakerAgent
- FacialEmotionsAgent
- PoseEmotionsAgent
- VocalEmotionsAgent

These agents have the task of extracting emotion from images and audio recordings and to combine all the information to decide the most probable emotion.

There are two other agents that can be used for CoppeliaSim simulation:

- VideoSimulationManagerAgent
- AudioSimulationManagerAgent

They allow to easily manage CoppeliaSim robots, to record audio files and images and to perform face detection on images. CoppeliaSim management is provided by my ah hoc Python 3 library, BlocksBot.

Thanks to BlocksBot library, a developer can easily handle simulation objects:

- Take data and current state information from simulation objects (for example sensor values)
- Give commands to simulation objects

Finally, there are three Python 2 agents for real robot, Nao:

- ReactionNaoBotManager
- AudioNaoBotManager
- VideoNaoBotManager

`ReactionNaoBotManager` gets the result of decision, the probable current emotion, and properly react to it.

The other two agents have to collect images and audio recordings, similarly to `VideoSimulationManagerAgent` and `AudioSimulationManagerAgent`, but `AudioNaoBotManager` is still in laboratory test phase.

Developers can create real robot modules by expanding BlocksBot library Robot class, but with Nao it is not possible, because it supports Python 2 only.

BlocksBot Project is available on GitHub (see [5]).

3.5 Runners

`Runner` is the special agent I developed to start and stop all other agents, in order to have clean start and exit for these parallel daemon processes. `Runner` is configurable: developer can define which agents `Runner` has to start. BlockBot provides two Runners:

- Runner inside NaoBot folder is coded in Python 2, so can run on real robot, Nao
- Runner inside main project folder is coded in Python 3 and can run on computer

`NaoBot Runner` is based on Python os library, conversely `main project folder Runner` uses Python multiprocessing and subprocess library.

User can start and stop agents by giving "start" and "exit" commands via Redis, publishing the command on a dedicated channel, that is defined in Redis configuration file.

`NaoBot Runner` provide a "stop" command too, to temporarily stop all agents except for `Runner` itself. Contrary to the other one, `NaoBot Runner` can only start agents: NaoBot agents are always listening on command channel to detect if a stop command has been launched and eventually terminate. When en "exit" command occurs, `NaoBot Runner` publishes a "stop" command by itself on the proper command channel, that is the channel the other agents are listening on.

[5] `https://github.com/agnsal/BlocksBot`

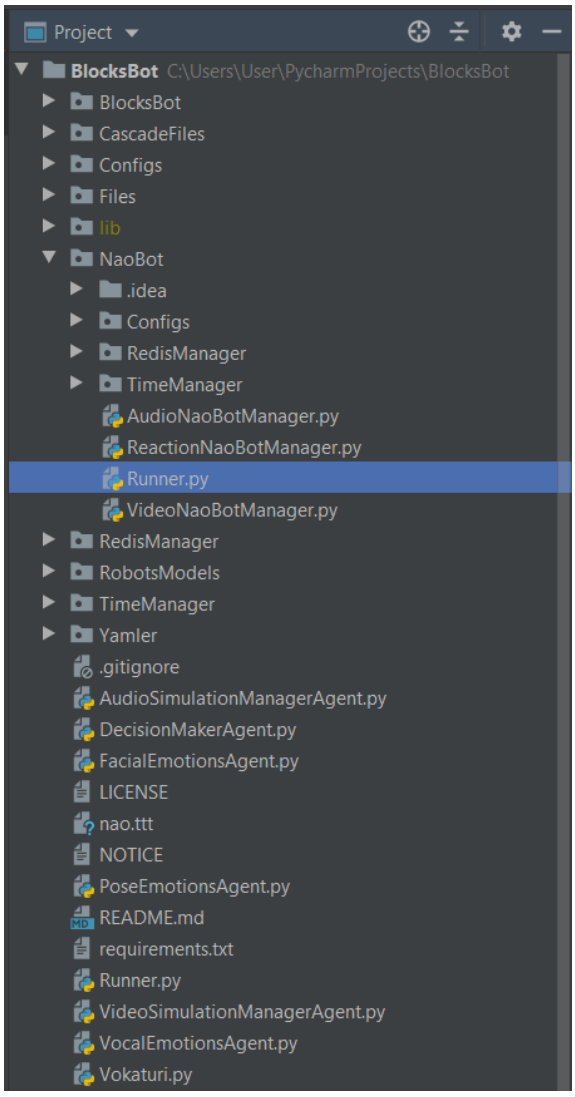


Figure 3.8: BlocksBot structure

3.6 Configurations

There are many configuration Yaml Files:

- DMAConfig for `DecisionMakerAgent`
- FacePlusPlusConfig for Face++ parameters
- RedisConfig for Redis
- RunnerConfig
- SimulationConfig for CoppeliaSim simulation

My FacePlusPlusConfig file contains my Face++ credentials to access their online services. To run the system on your machine, you have to create your own profile and add your credentials inside that file, at the following sections:

- API_KEY: '<your key>'
- API_SECRET: '<your password>'

RedisConfig contains the name of the used channels and queues and set names and fields. It contains Redis parameters too: host, port, database, password and data coding mode.

RunnerConfig contains the paths of the agents that `main project folder Runner` has to launch and eventually terminate.

SimulationConfig contains parameters used by Simulations agents.

It contains the path of the Json file where Nao body is described, congruently to its simulation representation. Robots description files are contained by RobotsModels folder. Robot body is described as a Json file, whose content is analogous to a Python dict. A robot body description is characterized by a set of sensors and a set of actuators. Each sensor or actuator is itself a set consisting of a name and a model.

A component name and model have to be coherent to the ones of its simulation representation, so two components cannot have the same name, because it is unique in simulation, but there can be many components with the same model.

NAO.json

```
{
  "sensors": {
    "HeadVision1": {"name": "vision_sensor1", "model": "vision_sensor"},
    "HeadVision2": {"name": "vision_sensor2", "model": "vision_sensor"},

    "LFootForce1": {"name": "NAO_LFsrFL", "model": "force_sensor"},
    "LFootForce2": {"name": "NAO_LFsrFR", "model": "force_sensor"},
    "LFootForce3": {"name": "NAO_LFsrRL", "model": "force_sensor"},
    "LFootForce4": {"name": "NAO_LFsrRR", "model": "force_sensor"},

    "RFootForce1": {"name": "NAO_RFsrFL", "model": "force_sensor"},
    "RFootForce2": {"name": "NAO_RFsrFR", "model": "force_sensor"},
    "RFootForce3": {"name": "NAO_RFsrRL", "model": "force_sensor"},
    "RFootForce4": {"name": "NAO_RFsrRR", "model": "force_sensor"}
  },

  "actuators": {
    "HeadYawJoint": {"name": "HeadYaw", "model": "joint"},
    "HeadPitchJoint": {"name": "HeadPitch", "model": "joint"},

    "LShoulderPitch": {"name": "LShoulderPitch3", "model": "joint"},
    "LShoulderRoll": {"name": "LShoulderRoll3", "model":  "joint"},
    "LElbowYaw": {"name": "LElbowYaw3", "model":  "joint"},
    "LElbowRoll": {"name": "LElbowRoll3", "model":  "joint"},
    "LWristYaw": {"name": "LWristYaw3", "model":  "joint"},

    "LThumbBase": {"name": "NAO_LThumbBase", "model":  "joint"},
    "LThumbMiddle": {"name": "Revolute_joint8", "model":  "joint"},
    "LLFingerBase": {"name": "NAO_LLFingerBase", "model":  "joint"},
    "LLFingerMiddle": {"name": "Revolute_joint12", "model":  "joint"},
    "LLFingerTop": {"name": "Revolute_joint14", "model":  "joint"},
    "LRFingerBase": {"name": "NAO_LRFingerBase", "model":  "joint"},
    "LRFingerMiddle": {"name": "Revolute_joint11", "model":  "joint"},
    "LRFingerTop": {"name": "Revolute_joint13", "model":  "joint"}
```

Figure 3.9: Nao Model extract

SimulationConfig contains also the path of the xml cascade file that is used by `VideoSimulationManagerAgent` OpenCV 2 Neural Network to locally make face detection, in fact this file in used by OpenCV (see [6]) to create a pre-trained Neural Net.

Xml cascade files are contained by CascadeFiles folder. In my case, I had to perform simulations and image recording with my webcam, so I needed only frontal face cascade file, but you can easily change it or add other Neural Networks to detect profile faces or other body parts.

I chose Yaml files because are human-readable and very fast to write, but Python 2 doesn't support them, so I had to crate a Python 2 package, called Configs, inside NaoBot folder for the real robot. Python 2 Configs contains all the configuration parameters needed by NaoBot agents, included Redis ones.

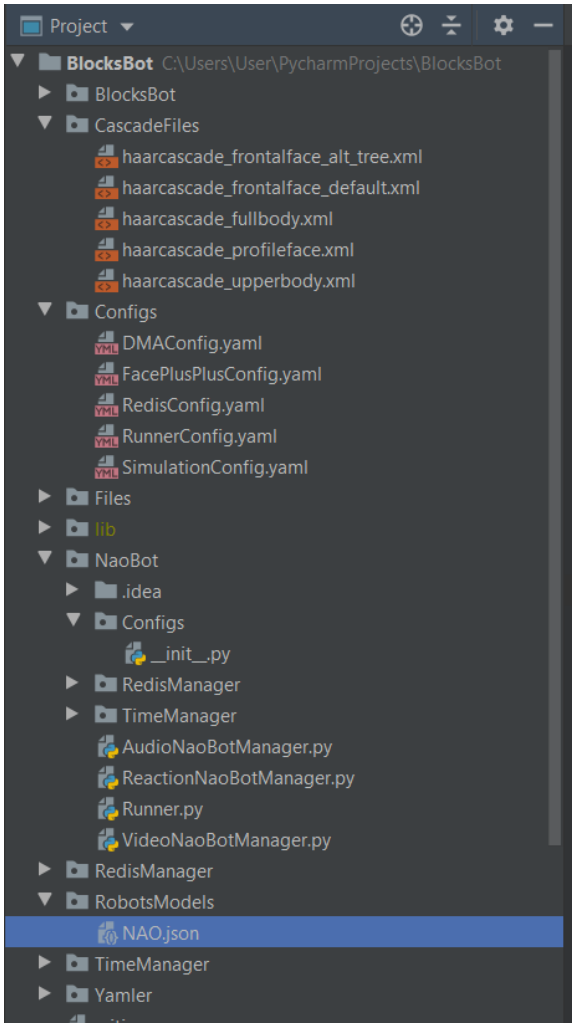


Figure 3.10: BlocksBot configuration files

[6] `https://opencv.org/`

3.7 BlocksBot Tools

For BlocksBot project I created many different tools with different tasks:

- BlocksBot Python 3 library
- RedisManager Python 2/3 library
- TimeManager Python 2/3 module
- Yamler Python 3 module

Code for real robot, Nao, like `NaoBot Runner`, is contained by NaoBot folder. BlocksBot library and Yamler module are not Python 2 backward compatible, so they cannot run on Nao. NaoBot folder contains:

- NaoBot agents
- RedisManager library
- TimeManager module
- NaoBot Configs module

BlocksBot library aim is to provide a general library to work both with real robots and with CoppeliaSim simulation. It gets a Json file describing a robot as input and to get the handles of the simulation representation of the robot. It allows to easily manage simulation objects, taking control of every robot sensor and actuator. It allows to manage other simulation objects too, like console windows, that can be created and populated with new data.

CoppeliaSim simulations are saved on ttt files. BlocksBot simulation is stored in nao.ttt file, that is inside main project folder and it can be runned with by double clicking on it. Obviously, in order to launch a CoppeliaSim simulation, first of all you have to install CoppeliaSim simulator on your computer (you can download it for free, see [7]).

RedisManager is a Python 2/3 library to easily manage Redis environment and to perform Redis calls. It is based on Python Redis library.

TimeManager is a simple Python 2/3 module, based on Python library Time, to easily create timestamps. I use these timestamps for Redis inputs. Timestamps unity corresponds to a tenth of a second.

Yamler is a simple Python 3 module that allow to easily read Yaml configuration files and get their content as a Python dict.

3.8 VideoSimulationManagerAgent and AudioSimulationManagerAgent

`VideoSimulationManagerAgent` is a process with the following tasks:

- Connect with nao.ttt simulation
- Put simulation Nao in the proper posture, that is the stand posture (see [8])
- Open the webcam to get user images
- Detect user face
- Periodically store images on Redis
- Publish a notification on the proper channel when a new image is stored on Redis
- Analyze user movements

[7] `https://www.coppeliarobotics.com/`

[8] `http://doc.aldebaran.com/2-1/family/robots/postures_robot.html`

- Make a plan based on user movements and then send the proper commands to simulation to make simulation Nao react to user
- Get probable current user emotion when produced by `DecisionManagerAgent` from Redis
- Write this emotion in a simulation window

This module is very important, considering free version of CoppeliaSim new limitations. In the past, V-REP, that is CoppeliaSim predecessor, allowed to directly process images inside simulations, performing many complex actions, like object tracking. Currently all these functions are no more available, so using an external real camera to get images and process them with an external script is the only possible solution for hard image processing. Another advantage of doing such a thing is that code is portable for real robots too. Furthermore, using a real camera instead of simulation cameras, images are so defined that you can make more efficient and realistic tests on image processing algorithms. For `VideoSimulationManager` I used OpenCV (see [9]), an open source, performing and widely used Python library.

`AudioSimulationManagerAgent` is a process with the following tasks:

- Periodically record sounds in wav format
- Store audio recordings on Redis
- Publish a notification on the proper Redis channel when a new audio recording is stored on Redis

Together with `VideoSimulationManagerAgent`, `AudioSimulationManager` works in simulation background. To try the system, you have to launch the nao.ttt simulation before `main folder Runner`.

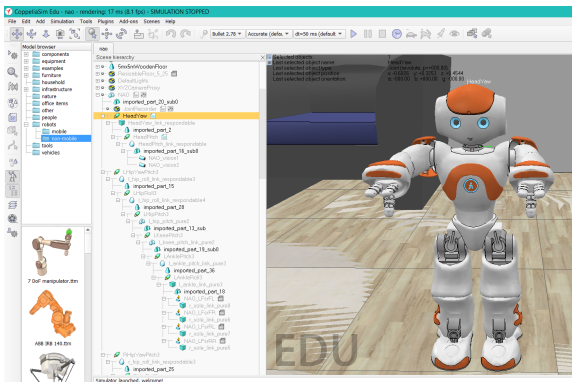


Figure 3.11: nao.ttt simulation

User can perceive simulation experience like a video-call with Nao. Nao moves its neck to always look at the moving user. At the same time, user probable current emotion is printed on top of a dedicated simulation window.

`VideoNaoBotManager` and `AudioNaoBotManager` are very similar to `VideoSimulationManagerAgent` and `AudioSimulationManager`, in fact `VideoNaoBotManager` and `AudioNaoBotManager` are Python 2 `VideoSimulationManagerAgent` and `AudioSimulationManager` versions that can run on Nao and perform exactly the same actions.

Remember that each agent can run in a distributed mode, that is on a different machine.

To distribute a module, it is needed to:

- Move it with every needed tools, for example RedisManager and `Runner`
- Change RunnerConfig agent paths
- Properly modify Redis configuration parameters

[9] `https://opencv.org/`

3.9 FacialEmotionsAgent, PoseEmotionsAgent and VocalEmotionsAgent

`FacialEmotionsAgent` and `PoseEmotionsAgent` have the following tasks:

- Wait for new images notifications
- Get each new image from Redis
- Analyze new image, using Face++ proper service
- Put analysis result in the proper format
- Store resulting data on Redis

The velocity of these two agents depends on the Internet and on Face++ servers. Usually, they can process around three images in a second.

`VocalEmotionsAgent` has the following tasks:

- Wait for new audio recordings notifications
- Get each new audio from Redis
- Analyze new audio recording, using Vokaturi library
- Put analysis result in the proper format
- Store resulting data on Redis
- Publish a notification on the proper Redis channel when a new vocal emotion result is stored on Redis

`VocalEmotionsAgent` is the slowest one to give its result. Actually it is not slow itself, but it has to wait until a new audio recording is available and this takes some seconds. In fact, capturing an image is much more faster than recording voice, so vocal emotion result is usually the last of all. For this reason, `VocalEmotionsAgents` is the one in charge of notifying when he is done processing.

3.10 DecisionMakerAgent

`DecisionMakerAgent` has the following tasks:

- Wait until `VocalEmotionsAgent` is done processing
- Retrieve `VocalEmotionsAgent` result from Redis
- Take all `FacialEmotionsAgent` results from the proper Redis queue
- Calculate the average of `FacialEmotionsAgents` results
- Properly combine that average result with `VocalEmotionsAgent` result to obtain a unique result
- Take all attitudes, that are `PoseEmotionsAgent` results, from the proper Redis queue
- Calculate the average of `PoseEmotionsAgents` results
- Properly combine that last result with the previously obtained combination to have the final decision
- Store final decision on Redis
- Publish a notification on the proper Redis channel
- Delete old Redis data

The combination of `VocalEmotionsAgent` result and `FacialEmotionsAgent` average result is performed as a weighted average: they are multiplied by their own reliability factor. `FacialEmotionsAgents` in fact is more accurate than the other, so it has a reliability factor major than the other. These factors can be configured in DMAConfig file.

The second combination is carried out if the fist combination result has a difference between the two most probable emotions that is under a certain threshold, that can be configured in DMAConfig. That process is interesting, because it is performed via Datalog engine, that is embedded in Python PyDatalog library.

First of all, I had to define Datalog terms and to give emotion classification to Datalog engine:

- `pyDatalog.create_terms('positive, neutral, negative, firstEmo, secondEmo, poseAttitude, X, Y, A, takeDecision, D, H, I, K')`
- `pyDatalog.assert_fact('positive', 'happiness')`
- `pyDatalog.assert_fact('neutral', 'neutral')`
  `pyDatalog.assert_fact('negative', 'sadness')`
  `pyDatalog.assert_fact('negative', 'fear')`
  `pyDatalog.assert_fact('negative', 'angry')`

The folloging is Datalog script, that contains logic rules, used by engine with explanations:

- `firstEmo(X) <= firstAvgEmo(X)`

  X is a fist emotion if it occurs that X is a first average emotion
- `secondEmo(Y) <= secondAvgEmo(Y)`

  Y a the second emotion if it occurs that Y is a second average emotion
- `attitude(A) <= poseAttitude(A)`

  A is an attitude if it occurs that A is a pose attitude
- `takeDecision(X) <= firstEmo(X) & secondEmo(Y) & attitude(A) & neutral(Y) & neutral(A)`

  X is the decision to take if X is a first emotion, Y is a second emotion, A is an attitude, Y is neutral and A is neutral
- `takeDecision(Y) <= firstEmo(X) & secondEmo(Y) & attitude(A) & neutral(X) & neutral(A)`

  Y is the decision to take if X is a first emotion, Y is a second emotion, A is an attitude, X is neutral and A s neutral
- `takeDecision(X) <= firstEmo(X) & secondEmo(Y) & attitude(A) & neutral(X) & positive(Y) & negative(A)`

  X is the decision to take if X is a first emotion, Y is a second emotion, A is an attitude, X is neutral Y is positive and A is negative
- `takeDecision(X) <= firstEmo(X) & secondEmo(Y) & attitude(A) & neutral(X) & negative(Y) & positive(A)`

  X is the decision to take if X is a first emotion, Y is a second emotion, A is an attitude, X is neutral, Y is negative and A is positive
- `takeDecision(X) <= firstEmo(X) & secondEmo(Y) & attitude(A) & positive(X) & neutral(Y) & positive(A)`

  X is the decision to take if X is a first emotion, Y is a second emotion, A is an attitude, X is positive, Y is neutral and A is positive
- `takeDecision(X) <= firstEmo(X) & secondEmo(Y) & attitude(A) & negative(X) & neutral(Y) & negative(A)`

  X is the decision to take if X is a first emotion, Y is a second emotion, A is an attitude, X is negative, Y is neutral and A is negative

- `takeDecision(X) <= firstEmo(X) & secondEmo(Y) & attitude(A) & negative(X) & negative(Y)`

  X is the decision to take if X is a first emotion, Y is a second emotion, A is an attitude, X is negative and Y is negative

- `takeDecision(X) <= firstEmo(X) & secondEmo(Y) & attitude(A) & positive(X) & positive(Y)`

  X is the decision to take if X is a first emotion, Y is a second emotion, A is an attitude, X is positive and Y is positive

- `takeDecision(Y) <= firstEmo(X) & secondEmo(Y) & attitude(A) & neutral(X) & positive(Y) & positive(A)`

  Y is the decision to take if X is a first emotion, Y is a second emotion, A is an attitude, X is neutral, Y is positive and A is positive

- `takeDecision(Y) <= firstEmo(X) & secondEmo(Y) & attitude(A) & neutral(X) & negative(Y) & negative(A)`

  Y is the decision to take if X is a first emotion, Y is a second emotion, A is an attitude, X is neutral, Y is negative and A is negative

- `takeDecision(Y) <= firstEmo(X) & secondEmo(Y) & attitude(A) & positive(X) & neutral(Y) & negative(A)`

  Y is the decision to take if X is a first emotion, Y is a second emotion, A is an attitude, X is positive, Y is neutral and A is negative

- `takeDecision(Y) <= firstEmo(X) & secondEmo(Y) & attitude(A) & negative(X) & neutral(Y) & positive(A)`

  Y is the decision to take if X is a first emotion, Y is a second emotion, A is an attitude, X is negative, Y is neutral and A is positive

This logic can be easily written, modified and read using Datalog, but it would be an if-else nightmare using procedural programming. Datalog is a subset of Prolog and there are other more powerful Prolog implementations, like SWI-Prolog (see [10]), SICStus Prolog (see [11]) and, most of all, Qulog/TeleoR (see [12]). I opted for Datalog because it is embedded in Python, so it does not require additional software to be installed. This means that Datalog is very portable because it is compatible with every machine that can run Python. Furthermore, thanks to PyDatalog library, it is extremely easy to integrate it in every Python application.

Considering positive = +, negative = -, neutral = 0 and _ = general value, previous logic can be summarized by the following table:

1st emotion	2nd emotion	attitude	decision
_	0	0	_
0	_	0	_
0	+	-	0
0	-	+	0
+	0	+	+
-	0	-	-
-	-	_	-
+	+	_	+
0	+	+	+
0	-	-	-
+	0	-	0
-	0	+	0

In order to pass information to Datalog engine, `DecisionMakerAgent` has to assert new facts:

- `pyDatalog.assert_fact('firstAvgEmo', str(firstEmotion))`

[10] `https://www.swi-prolog.org/`
[11] `https://sicstus.sics.se/`
[12] `http://staff.itee.uq.edu.au/pjr/HomePages/QulogHome.html`

- `pyDatalog.assert_fact('secondAvgEmo', str(secondEmotion))`
- `pyDatalog.assert_fact('poseAttitude', str(attitude))`

Then, `DecisionMakerAgent` can query Datalog engine:

- `decision = str(pyDatalog.ask("takeDecision(D)"))`

Finally, the agent cleans Datalog engine memory, retracting old facts:

- `pyDatalog.retract_fact('firstEmo', str(firstEmotion))`
- `pyDatalog.retract_fact('secondEmo', str(secondEmotion))`
- `pyDatalog.retract_fact('poseAttitude', str(attitude))`

3.11 ReactionNaoBotManager

`ReactionNaoBotManager` has to:

- Wait for new decision
- Get current emotion from decision, that is stored in Redis
- Tell which is current probable emotion
- Properly react to that emotion

Nao has been designed as a robot working with people, so it is provided of many tools to facilitate developer task of making the robot nice at interacting with users. In developing Nao reaction, I used Nao gestures, that are provided by its animated speech feature (see [13]). Gestures are position that Nao can assume while talking. They imitate people movements during speech and are categorized according to sensation they want to inspire in the interlocutor.

`ReactionNaoBotManager` makes Nao react to emotions. Nao translate probable current emotion in Italian, because laboratory Nao is configured for Italian language, then tells the name of that emotion while performing a gesture. Happy gestures are associated to positive emotions, doubtful gestures to neutral emotion end sad gestures to negative emotions. There are many happy, doubtful and sad gestures and `ReactionNaoBotManager` takes a gesture from the proper group randomly. Gesture - emotion associations are defined in NaoBot Configs and can be easily changed.

[13] `http://doc.aldebaran.com/2-1/naoqi/audio/alanimatedspeech_advanced.html`

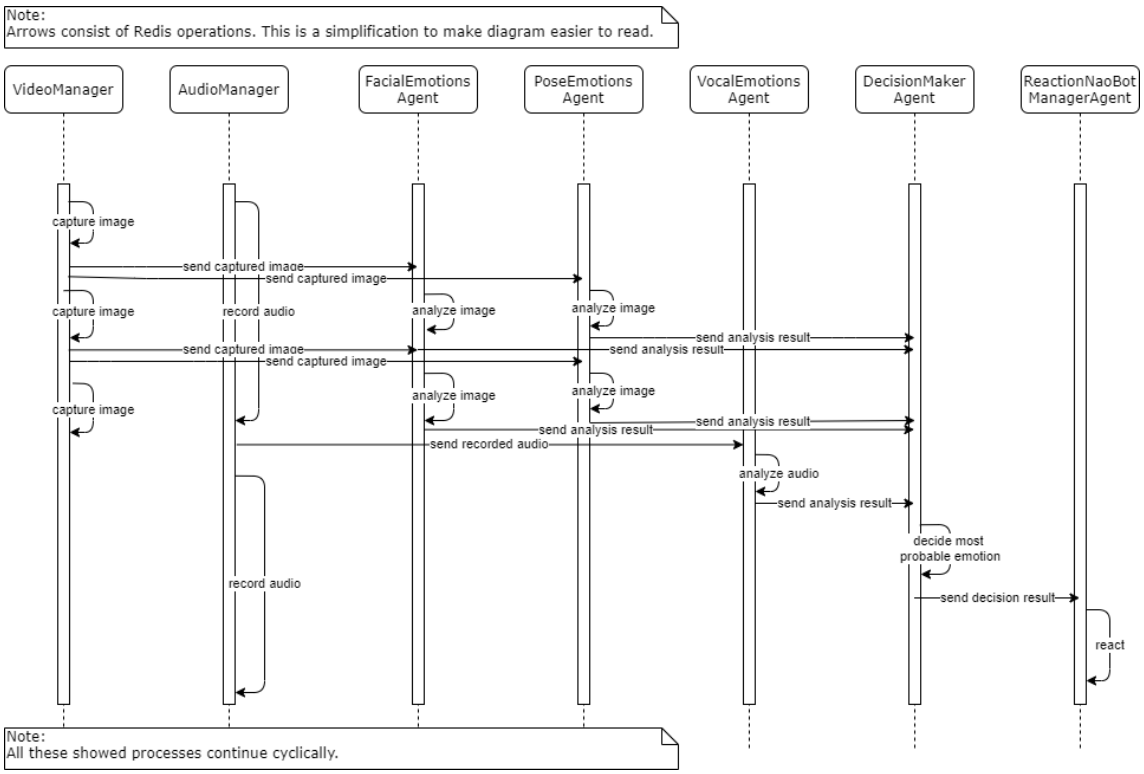


Figure 3.12: BlocksBot Sequence Diagram

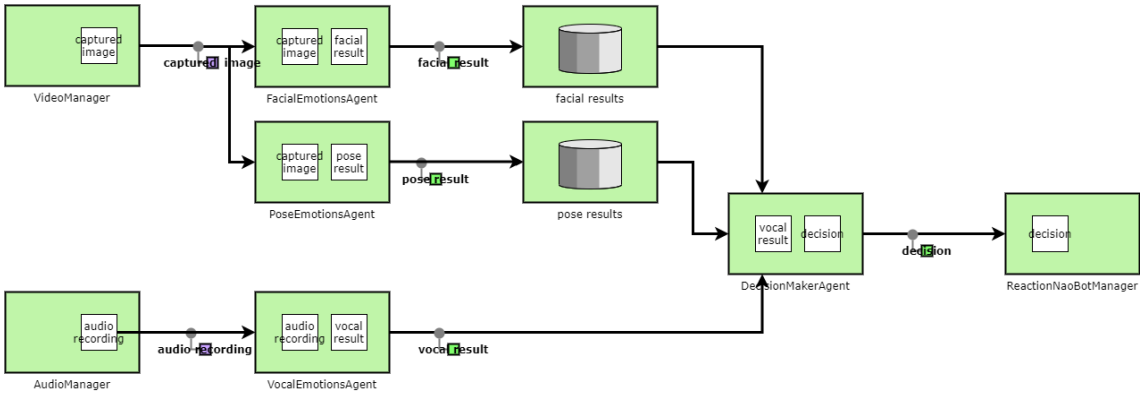


Figure 3.13: BlocksBot EIP Diagram

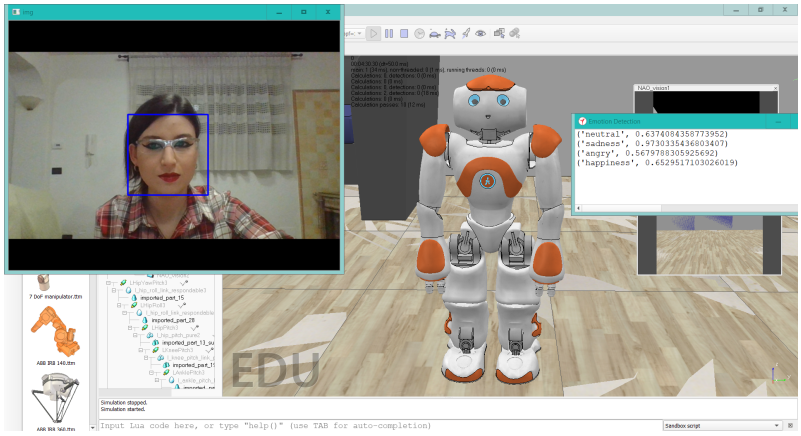


Figure 3.14: Running NaoBot with simulation

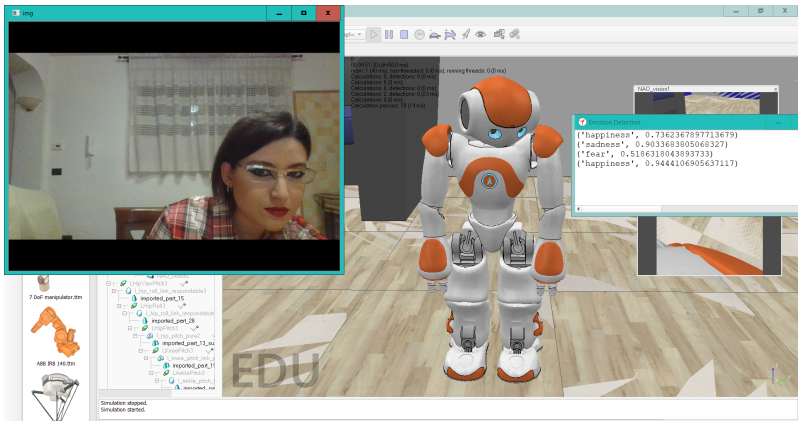


Figure 3.15: Running NaoBot with simulation

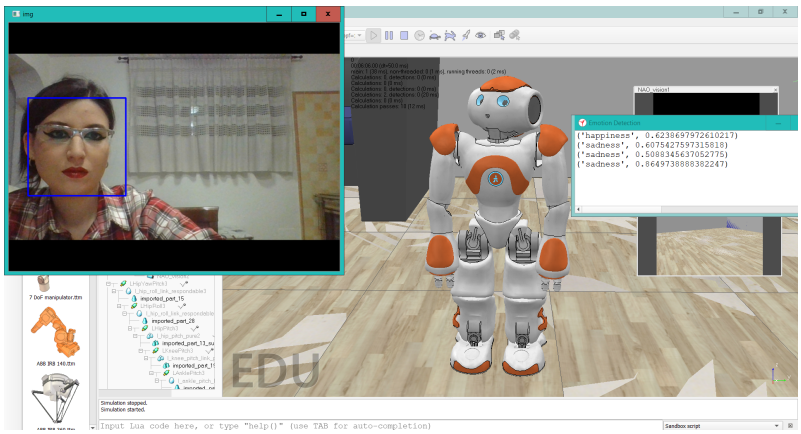


Figure 3.16: Running NaoBot with simulation

Chapter 4

Conclusion

In this final chapter, I am going to describe what's next for nascent BlocksBot project.

NaoBot is a very recent project, so this is its starting point. My initial aim was to provide instruments for empathic robotic, allowing developers to easily perform and integrate emotion detection and emulation to as much machines as possible. As every newborn project, there is still a lot of work to do on NaoBot.

Currently I am still working on NaoBot project with Professor De Gasperis, even if in these days we couldn't use the laboratory.

In the near future my objective is to expand NaoBot to improve its logic, in order to have a more performing emotion detection. The very next step could be to expand pose analysis to include other body parts.

However, first of all I have to perform a lot of tests on real Nao to validate its audio recording agent. Then I'd like to create new reactions emulating proper emotions.

Another important goal cold be to work with a Python 3 programmable robot, in order to expand NaoBot library RealRobot class.

Appendices

A Sitography of the State-of-the-art in Chapter 1

Here you can find all sitography of the State-of-the-art, described in Chapter 1.

- WABOT: http://www.humanoid.waseda.ac.jp/booklet/kato_2.html
- Kismet: http://www.ai.mit.edu/projects/humanoid-robotics-group/kismet/kismet.html
- Emotion Expression Humanoid Robot: http://www.takanishi.mech.waseda.ac.jp/top/research/we/we-4rII/index.htm
- Flobi: http://aiweb.techfak.uni-bielefeld.de/flobi-head?utm_source=robots.ieee.org
- Kaspar: https://robots.ieee.org/robots/kaspar/?gallery=photo4
- Furhat: https://www.furhatrobotics.com/
- Picoh: https://www.kickstarter.com/projects/ohbot2/picoh-an-expressive-little-robot-head
- ASIMO: https://asimo.honda.com/
- AIBO: https://us.aibo.com/
- PaPeRo: https://www.necplatforms.co.jp/solution/papero_i/
- Nao: https://www.softbankrobotics.com/emea/en/nao
- Alpha2: https://www.robotshop.com/en/ubtech-robotics.html
- PARRY: https://phrasee.co/parry-the-a-i-chatterbot-from-1972/
- AIML: https://code.google.com/archive/p/aiml-en-us-foundation-alice/downloads
- Jabberwaky: http://www.jabberwacky.com/
- Alexa: https://developer.amazon.com/it-IT/alexa
- Google Assistant: https://assistant.google.com/
- Siri: https://www.apple.com/it/siri/
- Cortana: https://www.microsoft.com/it-it/windows/cortana
- BN-1: http://www.theoldrobots.com/bn-1.html
- Meow-Chi: http://www.theoldrobots.com/images64/Meow-Chi.pdf
- Poo-Chi: http://www.robotsandcomputers.com/robots/manuals/Poo-Chi.pdf
- Necoro: https://robotnews.wordpress.com/2007/04/01/necoro-the-lovely-cat/
- I-Cybie: http://www.theoldrobots.com/icybie.html

- Furby: https://furby.hasbro.com/en-us
- PARO: http://www.parorobots.com/
- Pepper: https://www.softbankrobotics.com/emea/en/pepper
- Dinsow: https://www.dinsow.com/about.html
- My Friend Cayla: https://www.genesis-toys.com/my-friend-cayla
- My Friend Cayls case: https://www.bbc.com/news/world-europe-39002142

B Useful links

Here you can find all the useful links related to my projects.

- BlocksBot repository: `https://github.com/agnsal/BlocksBot`
- Redis: `https://redis.io/`
- CoppeliaSim: `https://www.coppeliarobotics.com/`
- Face++: `https://www.faceplusplus.com/`
- Vokaturi: `https://vokaturi.com/`
- OpenCV: `https://opencv.org/`
- Nao: `https://www.softbankrobotics.com/emea/en/nao`
- Nao predefined postures: `http://doc.aldebaran.com/2-1/family/robots/postures_robot.html`
- Nao animated speech: `http://doc.aldebaran.com/2-1/naoqi/audio/alanimatedspeech_advanced.html`
- Docker: `https://www.docker.com/`
- pyDatalog: `https://sites.google.com/site/pydatalog/home`
- SWI Prolog: `https://www.swi-prolog.org/`
- SICStus Prolog: `https://sicstus.sics.se/`
- Qulog/TeleoR: `http://staff.itee.uq.edu.au/pjr/HomePages/QulogHome.html`
- DALI repository: `https://github.com/AAAI-DISIM-UnivAQ/DALI`
- Keras: `https://keras.io/`
- Tensorflow: `https://www.tensorflow.org/`
- ServerDALImas repository: `https://github.com/agnsal/ServerDALImas`
- ServerPROLOG repository: `https://github.com/agnsal/ServerPROLOG`
- ServerDALI repository: `https://github.com/agnsal/ServerDALI`
- docker-ServerDALI repository: `https://github.com/agnsal/docker-ServerDALI`
- KOINE-DALI repository: `https://github.com/agnsal/KOINE-DALI`
- Redis2LINDA-stringESE repository: `https://github.com/agnsal/Redis2LINDA-stringESE`
- NeuralMAS repository: `https://github.com/agnsal/NeuralMAS`
- Crazyflie 2.0: `https://www.bitcraze.io/crazyflie-2/`
- CrazyRedis repository: `https://github.com/agnsal/CrazyRedis`
- LoraRobotRescue repository: `https://github.com/agnsal/LoraRobotRescue`
- prothonics-vrep repository: `https://github.com/agnsal/prothonics-vrep`
- prothonics repository: `https://github.com/agnsal/prothonics`
- DR2 repository: `https://github.com/C-A-V-ROBOT/DR2`
- docker-IndigoROSdisPyPl repository: `https://github.com/agnsal/docker-IndigoROSdisPyPl`
- kobukiROSindigo repository: `https://github.com/agnsal/kobukiROSindigo`

Bibliography

[1] Maral Gurbanzade, Edwin van der Heide, and Joost Broekens. Trust me if you can.

[2] Natalia Esau, Bernd Kleinjohann, Lisa Kleinjohann, and Dirk Stichling. Mexi: Machine with emotionally extended intelligence. In *HIS*, pages 961–970, 2003.

[3] Karsten Berns and Jochen Hirth. Control of facial expressions of the humanoid robot head roman. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3119–3124. IEEE, 2006.

[4] Yutaka Kondo, Kentaro Takemura, Jun Takamatsu, and Tsukasa Ogasawara. A gesture-centric android system for multi-party human-robot interaction. *Journal of Human-Robot Interaction*, 2(1):133–151, 2013.

[5] Mauricio E Reyes, Ivan V Meza, and Luis A Pineda. Robotics facial expression of anger in collaborative human–robot interaction. *International Journal of Advanced Robotic Systems*, 16(1):1729881418817972, 2019.

[6] Joseph Weizenbaum. Eliza—a computer program for the study of natural language communication between man and machine. *Communications of the ACM*, 9(1):36–45, 1966.

[7] Richard S Wallace. The anatomy of alice. In *Parsing the Turing Test*, pages 181–210. Springer, 2009.

[8] Giovanni De Gasperis. Building an aiml chatter bot knowledge-base starting from a faq and a glossary. *Journal of e-Learning and Knowledge Society*, 6(2):75–83, 2010.

[9] Stefania Costantini, Giovanni De Gasperis, Valentina Pitoni, and Agnese Salutari. Dali: A multi agent system framework for the web, cognitive robotic and complex event processing. In *ICTCS/CILC*, pages 286–300, 2017.

[10] Stefania Costantini, Tania Di Mascio, Giovanni De Gasperis, Patrizio Migliarini, and Agnese Salutari. Proposal of a Empathic Multi-agent Robot Design based on Theory of Mind. `https://caesar2020.di.unito.it/pdf/cAESAR2020_paper_5.pdf`, 2020.

[11] Stefania Costantini, Giovanni De Gasperis, and Patrizio Migliarini. Multi-agent system engineering for emphatic human-robot interaction. In *2019 IEEE Second International Conference on Artificial Intelligence and Knowledge Engineering (AIKE)*, pages 36–42. IEEE, 2019.