POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Simulation of Emotional Behaviour in a Robot

TESI DI LAUREA MAGISTRALE IN
COMPUTER SCIENCE ENGINEERING - INGEGNERIA INFORMATICA

Author: Claudia Chiroli

Student ID: 940320
Advisor: Prof. Andrea Bonarini

Academic Year: 2020-21

Abstract

Human robot interaction is the field of robotics whose aim is to study how humans and robots can cooperate and interact with each other. To accomplish this goal a central factor is the capability of the robot to correctly interpret human emotions and react in a way that can be considered an appropriate answer.

To study how emotion expression can improve human-robot interaction a good test field is theatre. The aim of this work is to study how a simple robot can interact with a human actor in an improvisational context, communicating emotions in both directions.

Key-words: Autonomous Robot, Actor, Behaviour, Emotions, Improvisation, Theatre, Human-Robot Interaction, ROS

Abstract in lingua italiana

L'interazione uomo-robot è il campo della robotica il cui scopo è studiare come l'uomo e i robot possono cooperare e interagire tra loro. Per raggiungere questo obiettivo un fattore centrale è la capacità del robot di interpretare correttamente le emozioni umane e reagire in un modo che può essere considerato una risposta adeguata.

Per studiare come l'espressione delle emozioni può migliorare l'interazione uomo-robot un buon campo di prova è il teatro. Lo scopo di questo lavoro è quello di permettere l'interazione di un semplice robot con un attore umano in un contesto di improvvisazione, comunicando emozioni in entrambe le direzioni.

Parole chiave: Robot autonomo, Attore, Comportamento, Emozioni, Improvvisazione, Teatro, Interazioni umani-robot, ROS

Table of Contents

List of Figures

List of Tables

1. Introduction

Human robot interaction is the field of robotics whose aim is to study how humans and robots can cooperate and interact with each other. This field has become central with the evolution of robots that has driven them to cooperate with humans and therefore in some applications require them to communicate in a more “human" way, i.e. not only with text lines and computer-generated voice but also emphasizing the interaction by showing emotional expressions like humans do.

To study how emotion expression can improve human-robot interaction a good test field is theatre. Theatre is a context where emotions are scripted, and there are not as many unexpected events as in the real world, so it is ideal to focus on emotions expression. Despite this, theatre is not completely detached from the real world, because it must be truthful to be effective.

The purpose of this work is to enable a robot to interact with a human actor, communicating emotions in both directions. We did not aim to provide the robot with a real “intelligent” behaviour (that is, make him capable of analyse a situation in detail and decide its future course of action on the basis of the evolution of the context). However, the goal is to make the robot react in an appropriate way, communicating an emotion consistent with the actor’s one; moreover, this reaction can be associated to a previously defined “personality” of the robot. The expectation is to obtain a sort of chained interaction evolving in a context that is built by the interaction itself between actor and robot, in a sort of improvised scene.

1.1 Brief Description of the Work

The capability of a robot to express emotions has been explored for example in the robot-cushion developed by dr. Julian Fernandez and prof. Andrea Bonarini in 2016 [30]. Their aim was to study how the different movements of the robot could be combined to express recognizable emotions. From their studies it emerged that even with a few degrees of freedom it is possible to make people recognize some emotions, thanks to the skilful modulation of movement speed and placing the robot in a context.

This concept has been further expanded in the graduation thesis developed by L. Bonetti under the inspiration and guidance of prof. A. Bonarini [28]. This thesis, building upon the foundation of a robot already developed at AIRlab (see the “Triskar

One" description in section 4.1 below), has implemented the possibility for the robot to act a predefined "script" structured as a sequence of actions triggered by a small set of conditions.

The actions included some basic abilities such as:

- moving the robot around the lab
- moving robot's body
- moving robot's eyes

In this way it has been possible to realize brief "scenes" acted by the robot with a human actor, even if the robot didn't really react to the behaviour of the actor but it was only "synchronized" with him.

Proceeding along this line of development, our goal has been to expand this architecture to allow an "improvised" interaction with the human actor. The recognition of actor's attitude is the object the graduation thesis developed by a colleague student (L. Farinelli); the result of this processing is passed to our software which proceeds to elaborate the emotional reaction of the robot and its expression by means of the same actions listed above (navigating around, moving eyes etc.).

Our study has been developed according to the following main axes:

- identify concepts as "emotions", personality" and "scene action" in a way sufficiently precise and formalized to allow the implementation of their expression by means of algorithms;
- define a "behavioral model" where the robot reacts to the emotions expressed by a human actor, autonomously performing a motion- and emotion-based effective improvisation
- actually implement these algorithms and this model on the robotic platform and verify the results obtained.

1.2 Structure of the Thesis

This document is structured as follows:

- Chapter 1 (this chapter) contains a general introduction defining the scope of our work and a general outline of the content of the thesis.
- Chapter 2 summarizes the theoretical framework of our study, i.e. the background upon which our work has been performed.
- Chapter 3 outlines the conceptual approach that we follow to reach our objectives.

- Chapter 4 describes the hardware platform of the robot used in the frame of this study. This platform was already available at the beginning of the study and has not been modified.
- Chapter 5 describes the software architecture developed in the course of our work. Some modules of this architecture have been reused from previous projects (in particular the modules in charge of controlling the robot's navigation and movements).
- Chapter 6 contains a description of the trials performed at the end of the study.
- Chapter 7 summarizes the conclusions we have reached and outlines some hypothesis for further developments of the product realized in this thesis.

2. Context Of The Study

As explained in the Introduction, our objective is to allow the interaction of a robot with a human actor in a way which is not merely predefined in a script but can be "improvised", i.e. it changes its behaviour according to the attitude of the actor.

To reach this goal, our first task has been of course to examine all what has been done in the past on matters concerning our study. In particular, we had to assess:

1) The way in which concepts like "behaviour", "emotions" and "personality" has been scientifically defined and analysed, so that we could translate these concepts from a "psychological" and "philosophical" level to a formal level which can be implemented in a software system.
2) The research performed and the results obtained in the field of robots specifically designed to interact with people.

2.1 Theoretical Foundations of Behaviour Modelling

2.1.1 Emotions

First, we had to define the term "emotion" in a precise way, in order be able to work on their implementation in a formal system. Hereafter is a brief summary of the literature on the topic.

2.1.2 What is an emotion?

"Emotions are biological states driven by the nervous system and associated with thoughts, feelings, behavioural responses, and a degree of pleasure or displeasure. Emotions are often intertwined with mood, temperament, personality, disposition, creativity, or motivation. In fact, an emotion involves processes of the mind, such as subjective experience, cognitive processes, expressive behavior, psycho-physiological changes, and instrumental behavior [38]". Living beings use emotions for self-preservation [61], like the feeling of being scared when someone is in danger: fear helps the individuals through tough situations and guides them in the process of choosing the correct countermeasure. Another example could be the disgusting taste that the spoiled food has, warning the individual about possible poisoned food. At the same time,

emotions are used as social regulators. In fact, humans use emotion to improve the interaction between each other, exchanging their thoughts and feelings not only by meaning of words, but also by expressions of their bodies, making the conversation more effective. Some species can also understand emotions and feelings of other species, for example dogs and humans are proved to communicate even without using words.

2.1.2.1.1 Discrete Model

In the history of Mankind many theories about the mind's mechanisms that drive emotions has been elaborated, and many systems have been proposed to classify them. For our work we decided to adopt the "Discrete Model" of emotions, which has also inspired a very successful animated movie ("Inside Out", produced in 2015 by Disney/Pixar) and the following set of products that are influencing emotion models in young generations in the Western countries.

The Discrete model is a theory developed by Paul Ekman and his colleagues in 1990 [37], who thought that all humans have an innate set of basic emotions that are cross-culturally recognizable. These basic emotions are described as "discrete" because they are believed to be distinguishable by an individual's facial expression and biological processes. They concluded that the six basic emotions are Anger, Disgust, Fear, Joy (Happiness), Sadness, and Surprise. Ekman explains that there are particular characteristics attached to each of these emotions, allowing them to be expressed at various degrees.

2.1.2.1.2 Emotions Expression: Laban Movement Analysis

Laban Movement Analysis [47][48] is a method and language for describing, visualizing, interpreting, and documenting human movement. Laban's original idea was to create a system that could allow different dancers to perform the same ballet, but now it is commonly used also in computer science, to teach computers to recognize human emotions. He divides the expression in 4 categories:

- **Body**

We define which parts of the body to move, in our case we have the eyes, the body, and the base (which moves on the stage). The choice of which parts to move is dictated by the stage needs.

- **Shape**

How the robot moves is affected by its construction, so apart from the provided movements it is impossible to introduce other ones unless further moving parts are introduced. The design and selection of the moving parts affect expression.

- **Space**

How the movements are connected with the environment is very important. A good interpretation of the robot is therefore linked to how he interacts with actors and props. For example, objects can be used as a hiding place (arousing an emotion of shyness or fear), it is also very effective to look at the interlocutor and this can be done by moving the eyes or the entire base.

- **Effort**

Effort is actually the robot's interpretation of its action and it is in turn divided into 4 parts:

  - *Weight*: It is the perception of the strength of the movement that can be expressed using the pauses between one movement and another. For example, long pauses can indicate some uncertainty, while a sequence of quick movements shows an excited or confident situation.
  - *Space*: Indicates how to rate the agent's focus on its target. For example, a movement directed towards the goal indicates a certain decision, introducing intermediate positions can cause more indecision.
  - *Timing*: The speed of each movement affects the viewer's perception of emotion. For example, happiness or anger are usually represented with rapid movements, while shame or sadness with slow movements.
  - *Flow*: Indicates the quality of the movements, whether rigid, “robotic" or more fluid, natural; for a robot this is strictly related to the control choices.

2.1.3 Personality Models

We all know that human behaviours are normally grouped in broad categories named “types of personality”. We analysed in details the two main systems used to define and classify personalities: the so-called “Big Five” model and the “Myers-Briggs Type Indicator” model.

2.1.3.1 Big Five Model

The Big Five Model, also known as the Five-Factor Model, is one of the most widely accepted personality theory held by psychologists today. The theory states that personality can be boiled down to five core factors, known by the acronym CANOE or OCEAN:

- openness to experience (inventive/curious vs. consistent/cautious)
- conscientiousness (efficient/organized vs. extravagant/careless)

- extraversion (outgoing/energetic vs. solitary/reserved)
- agreeableness (friendly/compassionate vs. critical/rational)
- neuroticism (sensitive/nervous vs. resilient/confident)

Unlike other trait theories that sort individuals into binary categories (i.e. introvert or extrovert), the Big Five Model asserts that each personality trait is a spectrum. Therefore, individuals are ranked on a scale between the two extreme ends. The following figure illustrates the Big Five Personality Scale and an example of the evaluation of a personality (blue crosses).:

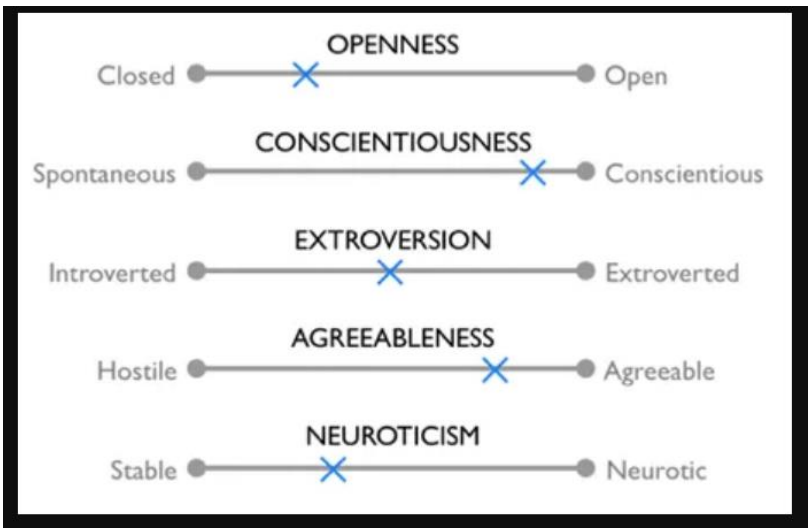

Figure 1: Big Five Personality Scale

For instance, when measuring Extraversion, one would not be classified as purely extroverted or introverted, but placed on a range determining the level of extraversion. By ranking individuals on each of these traits, it is possible to effectively evaluate individual differences in personality.

2.1.3.1.1 History and Background

The Big Five model resulted from the contributions of many independent researchers. Gordon Allport and Henry Odbert first formed a list of 4,500 terms relating to personality traits in 1936 [53]. Their work provided the foundation for other psychologists to begin determining the basic dimensions of personality.

In the 1940s, Raymond Cattell and his colleagues used factor analysis (a statistical method) to narrow down Allport's list to sixteen traits. However, numerous psychologists examined Cattell's list and found that it could be further reduced to five traits. Among these psychologists were Donald Fiske, Norman Smith and McCrae & Costa.

In particular, Lewis Goldberg advocated heavily for five primary factors of personality [59]. His work was expanded upon by McCrae & Costa, who confirmed the model's validity and provided the model used today: conscientiousness, agreeableness, neuroticism, openness to experience, and extraversion.

The model became known as the "Big Five" and has received much attention. It has been researched across many populations and cultures and continues to be the most widely accepted theory of personality today.

Each of the Big Five personality traits represents extremely broad categories which cover many personality-related terms. Each trait encompasses a multitude of other facets.

For example, the trait of Extraversion is a category that contains labels such as Gregariousness (sociable), Assertiveness (forceful), Activity (energetic), Excitement-seeking (adventurous), Positive emotions (enthusiastic), and Warmth (outgoing) [54].

Therefore, the Big Five while not completely exhaustive, cover virtually all personality-related terms.

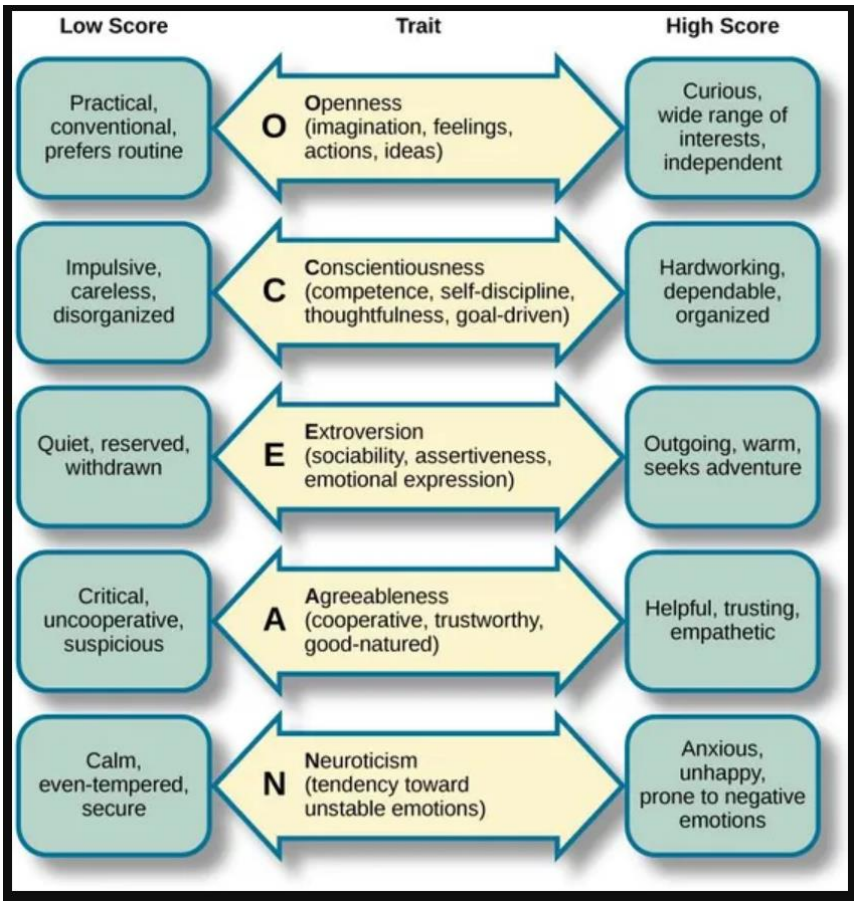

Figure 2: The Big Five Personality Traits

Another important aspect of the Big Five Model is its approach to evaluating personality. It focuses on conceptualizing traits as a spectrum rather than black-and-white categories, and recognizes that most individuals are not on the polar ends of the spectrum but rather somewhere in between.

2.1.3.1.2 The 5 Personality Traits

1) Conscientiousness

Conscientiousness describes a person's ability to regulate their impulse control in order to engage in goal-directed behaviours [57]. It measures elements such as control, inhibition, and persistency of behaviour.

Facets of conscientiousness include the following [54]:

High	Low
Competence	Incompetent
Organized	Disorganized
Dutifulness	Careless
Achievement striving	Procrastinates
Self-disciplined	Indiscipline
Deliberation	Impulsive

- *Conscientiousness vs. Lack of Direction*

Those who score high on conscientiousness can be described as organized, disciplined, detail-oriented, thoughtful, and careful. They also have good impulse control, which allows them to complete tasks and achieve goals.

Those who score low on conscientiousness may struggle with impulse control, leading to difficulty in completing tasks and fulfilling goals.

They tend to be more disorganized and may dislike too much structure. They may also engage in more impulsive and careless behaviour.

2) Agreeableness

Agreeableness refers to how people tend to treat relationships with others. Unlike extraversion which consists of the pursuit of relationships, agreeableness focuses on people's orientation and interactions with others [59].

Facets of agreeableness include the following [54]:

High	Low
Trust (forgiving)	Sceptical
Straightforwardness	Demanding
Altruism (enjoys helping)	Insults and belittles others
Compliance	Stubborn
Modesty	Show-off
Sympathetic	Unsympathetic
Empathy	Doesn't care about how other people feel

- *Agreeableness vs. Antagonism*

Those high in agreeableness can be described as soft-hearted, trusting, and well-liked. They are sensitive to the needs of others and are helpful and cooperative. People regard them as trustworthy and altruistic.

Those low in agreeableness may be perceived as suspicious, manipulative, and uncooperative. They may be antagonistic when interacting with others, making them less likely to be well-liked and trusted.

3) Extraversion

Extraversion reflects the tendency and intensity to which someone seeks interaction with their environment, particularly socially. It encompasses the comfort and assertiveness levels of people in social situations.

Additionally, it also reflects the sources from which someone draws energy.

Facets of extraversion include the following [54]:

High	Low
Sociable	Prefers solitude
Energized by social interaction	Fatigued by too much social interaction
Excitement-seeking	Reflective
Enjoys being the center of attention	Dislikes being the center of attention
Outgoing	Reserved

- *Extraversion vs. Introversion*

Those high on extraversion are generally assertive, sociable, fun-loving, and outgoing. They thrive in social situations and feel comfortable voicing their opinions. They tend to gain energy and become excited from being around others.

Those who score low in extraversion are often referred to as introverts. These people tend to be more reserved and quieter. They prefer listening to others rather than needing to be heard.

Introverts often need periods of solitude in order to regain energy as attending social events can be very tiring for them. Of importance to note is that introverts do not necessarily dislike social events, but instead find them tiring.

4) Openness to Experience

Openness to experience refers to one's willingness to try new things as well as engage in imaginative and intellectual activities. It includes the ability to "think outside of the box."

Facets of openness include the following [54]:

High	Low
Curious	Predictable
Imaginative	Not very imaginative
Creative	Dislikes change
Open to trying new things	Prefer routine
Unconventional	Traditional

- *Openness vs. Closedness to Experience*

Those who score high on openness to experience are perceived as creative and artistic. They prefer variety and value independence. They are curious about their surroundings and enjoy traveling and learning new things.

People who score low on openness to experience prefer routine. They are uncomfortable with change and trying new things, so they prefer the familiar over the unknown. As they are practical people, they often find it difficult to think creatively or abstractly.

5) Neuroticism

Neuroticism describes the overall emotional stability of an individual through how they perceive the world. It takes into account how likely a person is to interpret events as threatening or difficult.

It also includes one's propensity to experience negative emotions.

Facets of neuroticism include the following [54]:

High	Low
Anxious	Doesn't worry much
Angry hostility (irritable)	Calm
Experiences a lot of stress	Emotionally stable
Self-consciousness (shy)	Confident
Vulnerability	Resilient
Experiences dramatic shifts in mood	Rarely feels sad or depressed

- *Neuroticism vs. Emotional Stability*

Those who score high on neuroticism often feel anxious, insecure and self-pitying. They are often perceived as moody and irritable. They are prone to excessive sadness and low self-esteem.

Those who score low on neuroticism are more likely to calm, secure and self-satisfied. They are less likely to be perceived as anxious or moody. They are more likely to have high self-esteem and remain resilient.

2.1.3.1.3 Stability of the Traits

People's scores of the Big Five remain relatively stable for most of their life with some slight changes from childhood to adulthood. A study by Soto and John [62] attempted to track the developmental trends of the Big Five traits.

They found that overall agreeableness and conscientiousness increased with age. There was no significant trend for extraversion overall although gregariousness decreased and assertiveness increased.

Openness to experience and neuroticism decreased slightly from adolescence to middle adulthood. The researchers concluded that there were more significant trends in specific facets (i.e. adventurousness and depression) rather than in the Big Five traits overall.

2.1.3.1.4 Factors that Influence the Big 5

Like with all theories of personality, the Big Five is influenced by both nature and nurture. Twin studies have found that the heritability (the amount of variance that can be attributed to genes) of the Big Five traits is 40-60%.

Jang et al. [63] conducted a study with 123 pairs of identical twins and 127 pairs of fraternal twins. They estimated the heritability of conscientiousness, agreeableness, neuroticism, openness to experience, and extraversion to be 44%, 41%, 41%, 61%, and 53%, respectively.

This finding was similar to the findings of another study, where the heritability of conscientiousness, agreeableness, neuroticism, openness to experience and extraversion were estimated to be 49%, 48%, 49%, 48%, and 50%, respectively [63].

Such twin studies demonstrate that the Big Five personality traits are significantly influenced by genes and that all five traits are equally heritable. Heritability for males and females do not seem to differ significantly.

2.1.3.1.5 Gender Differences

Differences in the Big Five personality traits between genders have been observed, but these differences are small compared to differences between individuals within the same gender.

Costa et al. [67] gathered data from over 23,000 men and women in 26 countries. They found that "gender differences are modest in magnitude, consistent with gender stereotypes, and replicable across cultures".

Women reported themselves to be higher in Neuroticism, Agreeableness, Warmth (a facet of Extraversion), and Openness to Feelings compared to men. Men reported themselves to be higher in Assertiveness (a facet of Extraversion) and Openness to Ideas.

Another interesting finding was that bigger gender differences were reported in Western, industrialized countries. Researchers proposed that the most plausible reason for this finding was attribution processes.

They surmised that actions of women in individualistic countries would be more likely to be attributed to her personality whereas actions of women in collectivistic countries would be more likely to be attributed to their compliance with gender role norms.

2.1.3.1.6 Behavioural Outcomes

- **Relationships**

In marriages where one partner scores lower than the other on agreeableness, stability, and openness, there is likely to be marital dissatisfaction [68].

- **Health**

Neuroticism seems to be a risk factor for many health problems, including depression, schizophrenia, diabetes, asthma, irritable bowel syndrome, and heart.

People high in neuroticism are particularly vulnerable to mood disorders such as depression. Low agreeableness has also been linked to higher chances of health problems [54].

There is evidence to suggest that conscientiousness is a protective factor against health diseases. People who score high in conscientiousness have been observed to have better health outcomes and longevity [54]

Researchers believe that such is due to conscientious people having regular and well-structured lives, as well as the impulse control to follow diets, treatment plans, etc.

- **Education**

A high score on conscientiousness predicts better high school and university grades. Contrarily, low agreeableness and low conscientiousness predict juvenile delinquency [54].

- **Work**

Conscientiousness is the strongest predictor of all five traits for job performance [54]. A high score of conscientiousness has been shown to relate to high work performance across all dimensions.

The other traits have been shown to predict more specific aspects of job performance. For instance, agreeableness and neuroticism predict better performance in jobs where teamwork is involved.

However, agreeableness is negatively related to individual proactivity. Openness to experience is positively related to individual proactivity but negatively related to team efficiency.

Extraversion is a predictor of leadership, as well as success in sales and management positions [54].

2.1.3.1.7 Limitations of the Big Five

- **Descriptor Rather Than a Theory**

The Big Five was developed to organize personality traits rather than as a comprehensive theory of personality.

Therefore, it is more descriptive than explanatory and does not fully account for differences between individuals [54]. It also does not sufficiently provide a causal reason for human behaviour.

- **Cross-Cultural Validity**

Although the Big Five has been tested in many countries and its existence is generally supported by findings, there have been some studies that do not support its model. Most previous studies have tested the presence of the Big Five in urbanized, literate populations.

A study by Gurven et al. [69] was the first to test the validity of the Big Five model in a largely illiterate, indigenous population in Bolivia. They administered a 44-item Big Five Inventory but found that the participants did not sort the items in consistency with the Big Five traits.

More research in illiterate and non-industrialized populations is needed to clarify such discrepancies.

- **Is 5 Really the Magic Number?**

A common criticism of the Big Five is that each trait is too broad. Although the Big Five is useful in terms of providing a rough overview of personality, more specific traits are required to be of use for predicting outcomes [54].

There is also an argument from psychologists that more than five traits are required to encompass the entirety of personality.

2.1.3.2 MBTI

The Myers-Briggs Type Indicator, also known as the MBTI [71], is a personality assessment developed in the 1940s by Katherine Briggs and Isabel Myers. Based on the theories of psychologist Carl Jung in his written work “Personality Types“, the mother-daughter team built upon Jung’s work and categorized human behaviour and personality into 16 distinct groups.

Figure 3: Myers-Briggs Personality Models

2.1.3.2.1 Opposed Personality Characteristics

The Myers-Briggs Type Indicator test is based on a core theory that explains how behavioural traits differ from person to person in different situations, thus resulting in various personality types. The MBTI personality test's concept is based on the assumption that the world's population is made up of these 16 different sorts of people. This is where the 16 MBTI Personality Types originated. These 16 personality types in the Myers-Briggs personality test are divided into pairs of opposed personality functions.

The four pairs of opposed personalities are:

- ✓ Extraversion (E) - Introversion (I)
- ✓ Intuition (N) - Sensing (S)
- ✓ Thinking (T) - Feeling (F)
- ✓ Judging (J) - Perceiving (P)

In the following paragraphs we describe the meaning of each opposing pair.

- **Extraversion (E) – Introversion (I)**

The extraversion-introversion dichotomy was first explored by Jung in his theory of personality types as a way to describe how people respond and interact with the world around them. While these terms are familiar to most people, the way in which they are used in the MBTI differs somewhat from their popular usage.

Extraverts (also often spelled extroverts) are "outward-turning" and tend to be action-oriented, enjoy more frequent social interaction, and feel energized after spending time

with other people. Introverts are "inward-turning" and tend to be thought-oriented, enjoy deep and meaningful social interactions, and feel recharged after spending time alone.

We all exhibit extraversion and introversion to some degree, but most of us tend to have an overall preference for one or the other.

- **Sensing (S) – Intuition (N)**

This scale involves looking at how people gather information from the world around them. Just like with extraversion and introversion, all people spend some time sensing and intuiting depending on the situation. According to the MBTI, people tend to be dominant in one area or the other.

People who prefer sensing tend to pay a great deal of attention to reality, particularly to what they can learn from their own senses. They tend to focus on facts and details and enjoy getting hands-on experience. Those who prefer intuition pay more attention to things like patterns and impressions. They enjoy thinking about possibilities, imagining the future, and abstract theories.

- **Thinking (T) – Feeling (F)**

This scale focuses on how people make decisions based on the information that they gathered from their sensing or intuition functions. People who prefer thinking place a greater emphasis on facts and objective data.

They tend to be consistent, logical, and impersonal when weighing a decision. Those who prefer feeling are more likely to consider people and emotions when arriving at a conclusion.

- **Judging (J) – Perceiving (P)**

The final scale involves how people tend to deal with the outside world. Those who lean toward judging prefer structure and firm decisions. People who lean toward perceiving are more open, flexible, and adaptable. These two tendencies interact with the other scales.

Of course, all people at least spend some time engaged in extraverted activities. The judging-perceiving scale helps describe whether you behave like an extravert when you are taking in new information (sensing and intuiting) or when you are making decisions (thinking and feeling).

2.1.3.2.2 The 16 Personalities

According to the preferences shown, a personality can be classified in one of the following 16 types (each letter E-I, I-S, T-F, J-P stands for a distinct preference):

- **INFP - The Healer**

INFPs are imaginative idealists, guided by their own core values and beliefs. To a Healer, possibilities are paramount; the reality of the moment is only of passing concern. They see potential for a better future, and pursue truth and meaning with their own flair.

- **INTJ - The Mastermind**

INTJs are analytical problem-solvers, eager to improve systems and processes with their innovative ideas. They have a talent for seeing possibilities for improvement, whether at work, at home, or in themselves.

- **INFJ - The Counsellor**

INFJs are creative nurturers with a strong sense of personal integrity and a drive to help others realize their potential. Creative and dedicated, they have a talent for helping others with original solutions to their personal challenges.

- **INTP - The Architect**

INTPs are philosophical innovators, fascinated by logical analysis, systems, and design. They are preoccupied with theory, and search for the universal law behind everything they see. They want to understand the unifying themes of life, in all their complexity.

- **ENFP - The Champion**

ENFPs are people-centred creators with a focus on possibilities and a contagious enthusiasm for new ideas, people and activities. Energetic, warm, and passionate, ENFPs love to help other people explore their creative potential.

- **ENTJ - The Commander**

ENTJs are strategic leaders, motivated to organize change. They are quick to see inefficiency and conceptualize new solutions, and enjoy developing long-range plans to accomplish their vision. They excel at logical reasoning and are usually articulate and quick-witted.

- **ENTP - The Visionary**

ENTPs are inspired innovators, motivated to find new solutions to intellectually challenging problems. They are curious and clever, and seek to comprehend the people, systems, and principles that surround them.

- **ENFJ - The Teacher**

ENFJs are idealist organizers, driven to implement their vision of what is best for humanity. They often act as catalysts for human growth because of their ability to see potential in other people and their charisma in persuading others to their ideas.

- **ISFJ - The Protector**

ISFJs are industrious caretakers, loyal to traditions and organizations. They are practical, compassionate, and caring, and are motivated to provide for others and protect them from the perils of life.

- **ISFP - The Composer**

ISFPs are gentle caretakers who live in the present moment and enjoy their surroundings with cheerful, low-key enthusiasm. They are flexible and spontaneous, and like to go with the flow to enjoy what life has to offer.

- **ISTJ - The Inspector**

ISTJs are responsible organizers, driven to create and enforce order within systems and institutions. They are neat and orderly, inside and out, and tend to have a procedure for everything they do.

- **ISTP - The Craftsperson**

ISTPs are observant artisans with an understanding of mechanics and an interest in troubleshooting. They approach their environments with a flexible logic, looking for practical solutions to the problems at hand.

- **ESFJ - The Provider**

ESFJs are conscientious helpers, sensitive to the needs of others and energetically dedicated to their responsibilities. They are highly attuned to their emotional environment and attentive to both the feelings of others and the perception others have of them.

- **ESFP - The Performer**

ESFPs are vivacious entertainers who charm and engage those around them. They are spontaneous, energetic, and fun-loving, and take pleasure in the things around them: food, clothes, nature, animals, and especially people.

- **ESTJ - The Supervisor**

ESTJs are hardworking traditionalists, eager to take charge in organizing projects and people. Orderly, rule-abiding, and conscientious, ESTJs like to get things done, and tend to go about projects in a systematic, methodical way.

- **ESTP . The Dynamo**

ESTPs are energetic thrill seekers who are at their best when putting out fires, whether literal or metaphorical. They bring a sense of dynamic energy to their interactions with others and the world around them.

2.1.3.2.3 Frequency of Personality Types

According to recent surveys, the ten most common types of personalities are the following:

Personality Type	Global Population Frequency
ISFJ	13.8%
ESFJ	12.3%
ISTJ	11.6%
ISFP	8.8%
ESTJ	8.7%
ESFP	8.5%
ENFP	8.1%
ISTP	5.4%
INFP	4.4%
ESTP	4.3%

2.2 State Of The Art

We present hereafter a set of important developments in the field of our interest, which have contributed to inspire our work.

2.2.1 Emotional Robots

- **Blossom**

Blossom [60] is one of the most recent open-source social robotics platform. The objective of the associated research was to build a puppet-robot easy to assemble and program, able to express emotion with few D.O.F, in fact it is moved by only 2 motors. It is built with tensile mechanisms, elastic components, and a soft exterior cover attached loosely to the body. It is controlled by a raspberry pi.

The result is an accessible and customizable social robot for researchers and children educators. Software is based on Wizard-of-Oz framework [42] which provides easy accessibility to the robot's code via http, it is multi-platform and easy to program, but it has some limitations in things you can do with. Gestures are programmed with an app for smartphone, which uses the accelerometer to detect human body motions and translates to robot's one. Exterior of the robot is in textile and can be easily replaced and personalized.

It lacks movement on the ground, because it does not have wheels and cannot interact with people because it does not have sensors.

Figure 4: Inside Blossom

Figure 5: Possible external aspects of Blossom

- **iCat**

iCat [34] is a robot able to express emotions developed by Philips. Its face is shaped as a cat with movable parts to show some expressions. A camera mounted on the robot detects human and expressions which are reproduced on the face of the robot. Its scope is the study of the motion coordination and the detection of human expressions, but it has no other possible uses outside the laboratory.

Figure 6: iCat

- **Keepon**

Keepon [3] is a social robot designed for interaction and research with children, developed by Hideki Kozima. Keepon has four motors, a rubber skin, two cameras in its eyes, and a microphone in its nose. Keepon has been used to study the underlying mechanisms of social communication. Its simple appearance and behaviour are intended to help children, even those with developmental disorders such as autism, to understand its attentive and emotive actions. The robot, usually under the control of a remote operator, has interacted with children in schools and remedial centers, as well as with general public and a simplified version also reached the market as a dancing toy.

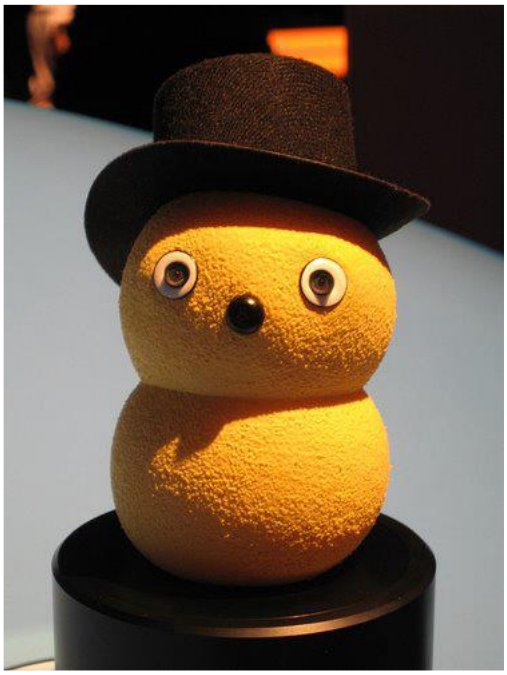

Figure 7: Keepon

Those who score low on conscientiousness may struggle with impulse control, leading to difficulty in completing tasks and fulfilling goals.

2.2.2 Theatrical Robots

There are many different styles of theatrical plays, different genres, different ways to act and communicate emotions. Similarly to the ”human” world, there are many different proposals of theatrical robots. The common thread of all these kinds of robots is that the robot has to perform a play, express concepts that the writer of the opera wants to underline, and amaze the public with its performance. Even though theatre is often associated with liveliness and physical presence, it is essential to put this assumption into perspective and realize that nonhuman agents (animals, objects, things, images) have been playing a more important role in theatre than the anthropocentric approach suggests. Replacing, representing, indicating the human presence with nonhuman agents on stage has a long tradition. Amongst other types of nonhuman technology, robot/android theatre performances are closely related to puppets. We report here below some examples of theatrical robots.

- **Humanoid robots**

Some researchers studied a way to build robotic actors very similar to humans. One of the masters of this trend is Hiroshi Ishiguro [52]. His studies are mainly focused on building robots that are as similar as possible to a person, but he also built some robots

able to perform in theatrical plays. In the show he wrote with the collaboration of the Japanese director Oriza Hirata [56] ”The Metamorphosis“ by Franz Kafka [46] was presented, but the main character Gregor Samsa was transformed into a robot instead of a cockroach. “Through creating performances featuring robots, Hirata and Ishiguro don’t attempt to define what a robot is, but what human is: whether a “core,” an essential quality, by which it is possible to define ”human,“ exists. In this concept, the shape of the humanoid robot is considered as a shell.

The activity of the robots is integrated into the performative event in three different ways: by software (by pre-recording the gesture sequences of a human actor), by pre-programming, or by remote control. As the robots appearing in Hirata’s shows don’t have artificial intelligence, they are not able to solve unexpected situations; therefore lack the ability of improvisational cooperation and creative activity. It means that the performative process is a one-way action: the robot executes the programmed sequence, the actor adapts, adjusts, reacts (and makes mistakes).”

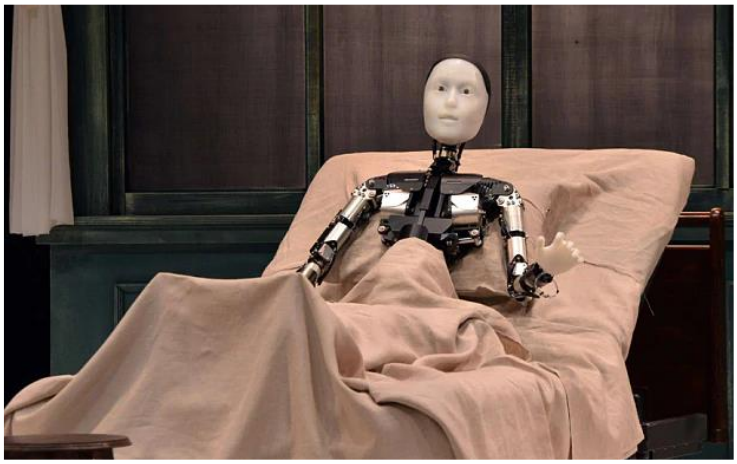

Figure 8: Robot performing ”The Metamorphosis”

The University of Taiwan developed a humanoid robot able to reproduce facial expressions and body expressions.

They developed software to perfectly mimic human movements observed with a camera and a believable lip-sync. With this robot they performed a small play behaving like a human. The result shows that even if the public is amazed by the technological progress, is not really interested on the improvements in the performance the robot does w.r.t the human actor.

- **La commedia dell’arte**

“La commedia dell’arte” [64] is one of the first projects of theatrical robots. They represented a play called “Lazzo of the Statue” taking inspiration from the Italian Renaissance theatre (“La commedia dell’arte”) based on predefined characters called

"masks", for example Pantalone, Arlecchino, Balanzone and so on. In this play Arlecchino pretends to be a statue who is being placed by the other players. Whenever they have their backs to him, he moves or changes position, forcing them to have to move him back in the desired position. Eventually, they get angry and Arlecchino runs away with the others in pursuit. They used 3 wheeled robots made by LEGO, they lack of sensors and, apart from Arlecchino, they totally lack of expressivity. Furthermore, the small size was not suitable for being seen from a distance as it could happen on a real stage.

Figure 9: Robot performing "Lazzo of the statue"

- **Robotic Therrarium**

One of the other first performances with a robot is the one written by the University of Cambridge [33] called "Robotic Therrarium", which consists in an anemone-shaped robot focused on performing actions based on the interaction with the public through a camera and some sensor. The problem of this robot is that it has a predefined character, and cannot interpret other characters, unlike a real actor.

- **Herb**

Herb [65] is a butler reprogrammed to perform a theatrical play, with the scope of substituting the human actor during the opera and understand how movement and prosodic speech can induce understanding of intention. "HERB is hardly an android: HERB lacks a really recognizable face, exhibits an arm motion range that is substantially non-anthropomorphic, and moves slowly with non-holonomic differential drive constraints." The usual HERB speech synthesizer is a general purpose text-to-speech product, modified with phonemes taken from a real actor to sound better. An operator controls the robot from remote and gives commands with the right timing, so the robot is not autonomous. The results were positive, the play was very appreciated from the public from both artistic and technical points of view.

It was only criticized because there were body parts not exploited (like the wheels).

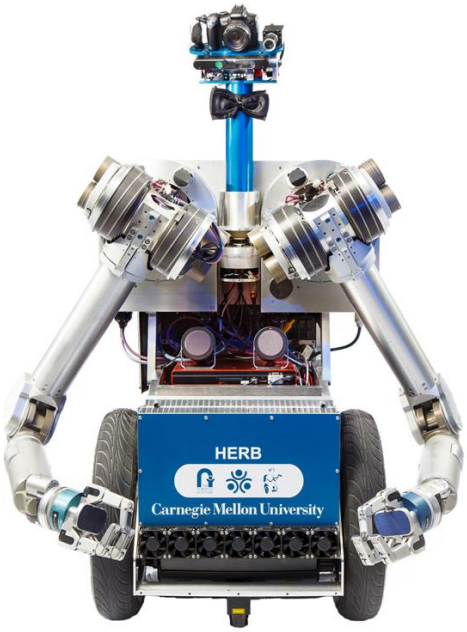

Figure 10: Herb

- **AUR**

AUR [43] is a robot-lamp inspired to the Pixar's logo. It can move its arm and change color and intensity of the light. The robot has a script divided in "beats" with actions to play, the software is in charge of executing the sequence of beats without an evident separation. The robot, thanks to a camera, keeps the eye contact with its human companion on stage. The timing of the action is given by an operator who decides when to start a beat, the operator also controls the status of the robot through a graphic interface and, in case of fault or emergency, can take control of any part of the robot. The actors who acted with the robot referred: "It seems to me that it was treated more like a toy (unfortunately), but then both me and the audience were like 'Wow, it's actually reacting as a human being' ". So the objective of building a robot able to act and not just considered as a prop was achieved. To improve the robot it was suggested to make it move around and be more autonomous, especially for what concerns timing.

Figure 11: AUR

- **Roboscopie**

Roboscopie [55] is a project that involves PR2, a non-anthropomorphic robot. The robot can move and localize itself into a static map of the stage. The task of the robot is to play with some object on the stage which recognizes thanks to a barcode. The robot is in part autonomous but it needs a human to take the object and make the correct flow of actions.

Figure 12: PR2

2.2.3 Humanoid Robots

- **NAO**

Nao [1] is a humanoid robot developed for many different uses. Many studies and projects in the emotion field used NAO for demonstrations. For example [40] NAO was used as a Stand-Up-Comedian. The interesting part of this study regards the decision of when it is the right moment to make a joke and when it is time to let the public laugh or applaud. The general problem of these multi-functional robots (another example could be Aibo [2]) is that they are quite limited in some specific movements and sometimes they are programmed to produce stereotyped ones.

High expectations are created due to their bio-inspired form, but they are disregarded due to their limited ability to move, which is why people lose interest in this kind of robot.

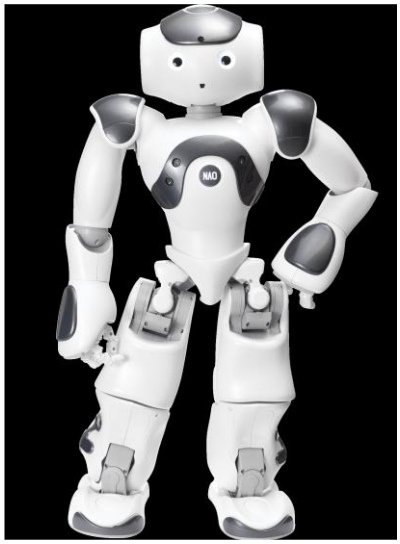

Figure 13: Nao

- **Simon**

Simon [41] is a humanoid robot built for studying how different shades of a movement can modify the perception of an emotion, in particular how exaggerated motion in experiments enhances the interaction through the benefits of increased engagement and perceived entertainment value. The results show that people tend to prefer cartoon like robots with exaggerated motion like Simon rather than ones that perfectly mimic real movements, because it is funnier and easier to understand.

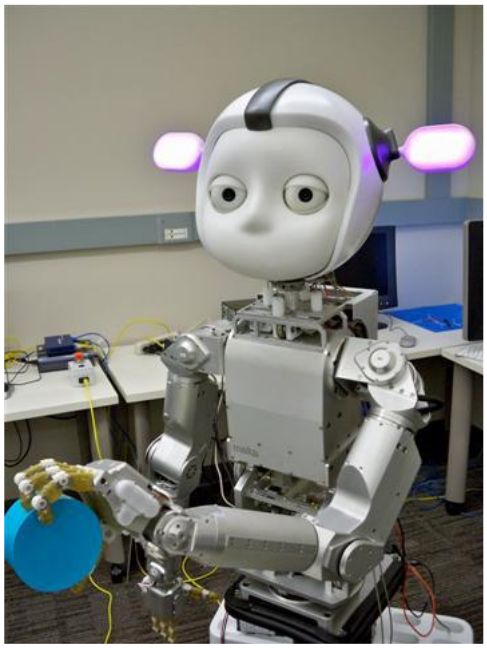

Figure 14: Simon

- **iCub**

The humanoid robot iCub [70] (I as in "I robot", Cub as in the man-cub from Kipling's Jungle Book), has been specifically designed by the Italian Institute of Technology (ITT) of Genova to support research in embodied artificial intelligence (AI). At 104 cm tall, the iCub has the size of a five-year-old child. It can crawl on all fours, walk and sit up to manipulate objects. Its hands have been designed to support sophisticate manipulation skills. The iCub is distributed as Open Source following the GPL/LGPL licenses and can now count on a worldwide community of enthusiastic developers. More than 30 robots have been built so far which are available in laboratories in Europe, US, Korea and Japan. It is one of the few platforms in the world with a sensitive full-body skin to deal with safe physical interaction with the environment.

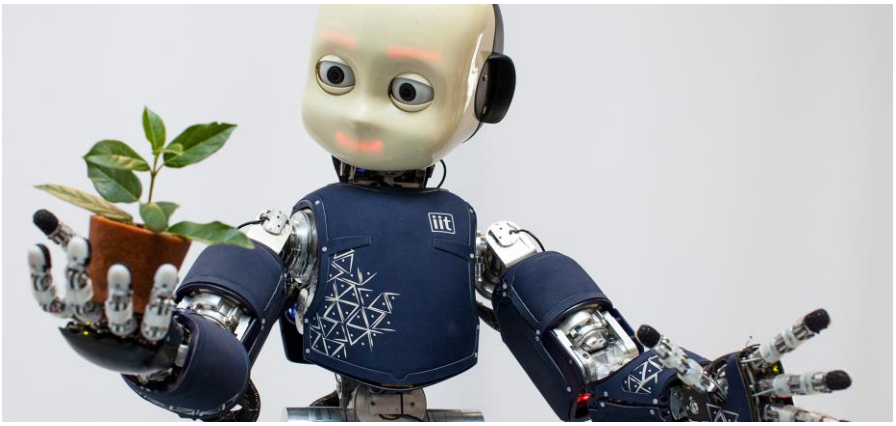

Figure 15: iCub

2.2.4 Improvisation

One of the abilities that an actor has to master is improvisation. There are studies to make a robot improvise [35] [58]. These are quite recent studies and few of this kind of robots have ever performed a show on stage. Improvisation requires a system able to understand the meaning of a play, the relations between the characters, the nature of the character the humour of his human companion and then program the robot to improvise actions and speak coherently with its character and the situation.

- **Shimon**

Shimon [44] is an interactive robotic marimba player that improvises in real-time while listening to, and building upon, a human pianist's performance. It has an improvisation system that uses a physical gesture framework based on the belief that musicianship is not merely a sequence of notes, but a choreography of movements. While part of the function of these movements is to produce musical sounds, they are also used to communicate with other band members and with the audience. The study concludes that adding these improvised movements improves the robot's performance and audience appreciation.

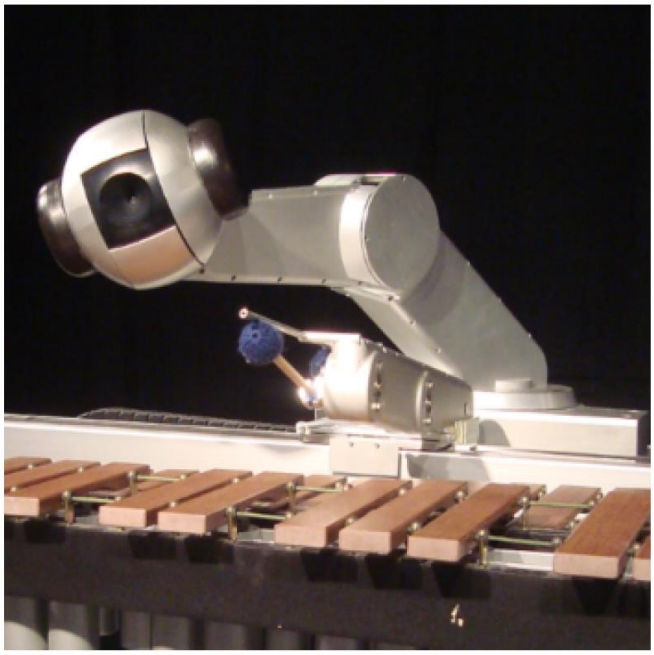

Figure 16: Shimon

3. Study Logic

3.1 Definition of a Behavioural Scheme

Our stated goal is to study how a simple robot can interact with a human actor in an improvisational context, communicating emotions in both directions. This means that the robot has in some way to "mimic" the human behaviour. But how do humans behave when interacting with each other ? Of course the answer to this question can be extremely complex, but our research lead us to identify a very simplified scheme:

1) First, we assess the "attitude" of the people in front of us. This assessment is:
    - almost always unconscious [29];
    - practically instantaneous [31] [39];
    - based essentially on non-verbal signals (movements, posture, face expressions, etc.) [45] [49];
    - aimed to "classify" the behaviour of our interlocutors within a defined set of possibilities (e.g. aggressive, amiable, cold, joyful, etc.) [50].
2) The behaviour of other people (as we have assessed it) causes inside us an emotional response: we can remain indifferent, become angry, feel happiness, and so on. Also this response is instantaneous and unconscious, i.e. it is not driven by any rational reasoning.

    It is worth noting that the very same actions and expressions can trigger different emotional responses in different subjects. The same view can cause anger in one person, sadness in another, disgust in yet another one, etc. This depends on what we call "personality", i.e. the distinctive traits that characterize each human being when interacting with other people.
3) On the basis of our new emotional state, and of the behaviour of the interlocutors, we actuate a reaction of some kind. This is the stage where our rational mind can intervene to moderate our reaction, usually driving it toward what is more opportune, socially acceptable, effective, considerate of the interlocutor, etc. However, our mediate answer comes later [51].

4) Our reaction is in turn assessed and processed by the other people (as we did in points 1, 2 and 3 above), giving rise to a chain of interaction loops which constitute our "social behaviour".

If we want to imitate human behaviour in a robot (of course within a controlled environment like a laboratory or a theatrical stage) we have therefore to:

1) Find a way to "recognize" the behaviour of the human interlocutor, which we assume to be a list of possible "scenic actions" which the actor can perform.
2) Model the "emotional life" of the robot, i.e. define a variable representing the robot's emotional state and implement an algorithm to make it evolve according to the recognized scenic actions. In doing this task we aspire to:
    - make the robot act with a certain level of "unpredictability", so that it does not appear to behave in a completely "mechanical" way;
    - endow the robot with a distinct "personality", which can be selected as an initial parameter of the performance and causes the robot to react in different ways to the same input stimuli (as discussed above).
3) Execute a "reaction" which shows the new emotional state of the robot.

In the next sections each one of the steps listed above is discussed in more detail.

3.2 Identification of Scenic Actions

In our context, "scenic actions" are defined as the complex of movements and facial expressions performed by a human actor with the aim to communicate his internal feelings and emotions in a way that can be recognized by our robot.

Recognizing a scenic action performed by the actor is a very complex task and required a specific study work which constitutes the subject of the graduation thesis of our colleague student L. Farinelli, who developed a software package dedicated to this purpose.

We take the output of this action recognition package as input to our software. The interface is implemented by means of the inter-process communications mechanisms provided by the Robot Operating System (ROS) middleware suite, as described below in the chapter presenting our Software Architecture.

The information acquired via this interface include:

A) a discrete categorization of the position of the actor ("front", "back", "left", "right")

B) the actual position of the actor, defined as a set of spatial coordinates

C) the scenic action identified by the recognition package, which can be one of the following:

1. Attack
2. Scolding
3. Intimidation
4. Grudge
5. Sharing Happiness
6. Happy person
7. Satisfaction
8. Sharing Fear
9. Running away
10. Sharing Sadness
11. Disappointment
12. Surprise
13. Disbelief (negative)
14. Astonishment
15. Leaving the Scene

3.3 Modeling the Robot's Emotional Life

3.3.1 Strategy

After having acquired the information concerning the "attitude" of the human actor, we have to model the second stage of the simplified behavioural scheme that we outlined in section 3.1 above, i.e. the change in our internal emotional state.

The strategy that we adopted to achieve this objective is based on the following principles:

- The robot's emotional evolution is modelled by means of the Finite-State Automaton (FSA) paradigm: each emotional state of the robot is a state of the FSA, and the transition between states is a function of the current state and the scenic action acquired in input.
- To simulate a certain level of "unpredictability" of the robot's behaviour, we decided to implement a *non-deterministic finite-state automaton* (NFA).
- As we stated above, the emotional response can differ from individual to individual according to their "personality". We therefore decided that the robot can have different personalities, selectable by means of a dedicated software parameter. Different personalities correspond to different emotional state transition functions.

3.3.2 Selection of Personality Model

To define the values that the "personality" parameter of the robot can have, we have to select a way for defining the set of possible personalities.

Both the personality models described in the previous chapter (Big Five and MBTI) are widely diffused and could be used for implementing a simulated emotional behaviour. However, we decided to adopt MBTI because of the following reasons:

1) MBTI intrinsically defines a discrete number of personality types: the 16 types resulting from the combination of the prevalent trait for each one of the 4 couples (E/I, N/S, T/F, J/P) outlined in section 2.1.2.2. Conversely, the "Big 5" model approach assigns the values for the 5 traits identified in section 2.1.2.1 (Openness, Conscientiousness, Extraversion, Agreeableness, Neuroticism) along axes in a continuous way; in this way it can generate a very high number of personality types.
2) Even if we had chosen to simplify the "Big Five" model by imposing a "binary" value for the possible traits (i.e., a personality can only be 100% Open, 0% Conscious etc.), the resulting number of personality types would be 2^5 = 32 types, a value double with respect to MBTI.

Because of these considerations, the MBTI model appears more suitable to be modelled in a software system and has therefore been selected for the current implementation.

3.3.3 Evolution of the Robot's Emotional State

As explained in the previous sections, we implemented the emotional response of the robot is modelled as follows:

- The different types of personality have been identified using the Myers-Briggs Indicator (MBTI) methodology described in section 2.1.2.2.
- For each one of the 16 identified personalities, a model to guide the evolution of emotional states has been assumed and implemented in our system.
- This implementation has been performed by means of Non-Deterministic Finite-Automaton (NFA), one NFA for each personality type.

From the formal point of view, the NFA implementing a personality type is defined as a quintuple (∑, S, s0, δ, F), where:

- ∑ (the *input alphabet*) is the set of "scenic actions" which can be performed by the human actor, as defined in section 3.2.
- S (the *set of states*) is the set of possible emotional conditions of the robot, as defined in section 2.1.1.1.2: Joy, Sadness, Fear, Anger, Surprise.
- s0 (the *initial state*) is one of the elements of S: it is the state in which the robot is when it starts the interactive session.
- δ: ∑ x S => $\mathcal{P}$(S) (the *state-transition function*) defines the way in which the emotional state changes according to the detected input. $\mathcal{P}$(S) is the power set of S; this of course means that from any given state the next state is not fixed but can be one in a range of possibilities. To each one of the possible transitions we have associated a "probability", which represent the likeliness to reach that emotional state.
- F (the set of *final states*) is empty: no specific termination of the robot-actor interactive session is defined; the termination of the robot-actor interactive session happens when one of the two performs a "Leave the scene" action, so the other does the same.

3.3.4 Performing Emotional State Evolution According to Personality

As explained in the previous section, for each different personality a specific state-transition table has been defined for the NFA defining the evolution of the robot's emotional state.

According to the NFA paradigm, given the current emotional state, more than one emotional state could be reached in consequence of the same scenic action recognized from the robot. To each possible transition, a probability has been associated,

representing the likeliness of the emotional response according to the defined personality type.

In our implementation the NFAs have been defined heuristically, but they could be optimized by learning the transition probabilities (for instance with a Reinforcement Learning algorithm based both on the attendance reaction or the quality of improvisation).

For this reason our SW has been developed in a completely parametric way: all the transition functions (and the related robot's reactions defined in section 3.4.1 below) are defined by means of external files written such as JSON or CSV that can easily be modified with an editor or with ad-hoc programs.

As an example, the state-transition diagram defining the "Architect (INTJ)" personality is show in the following figure:

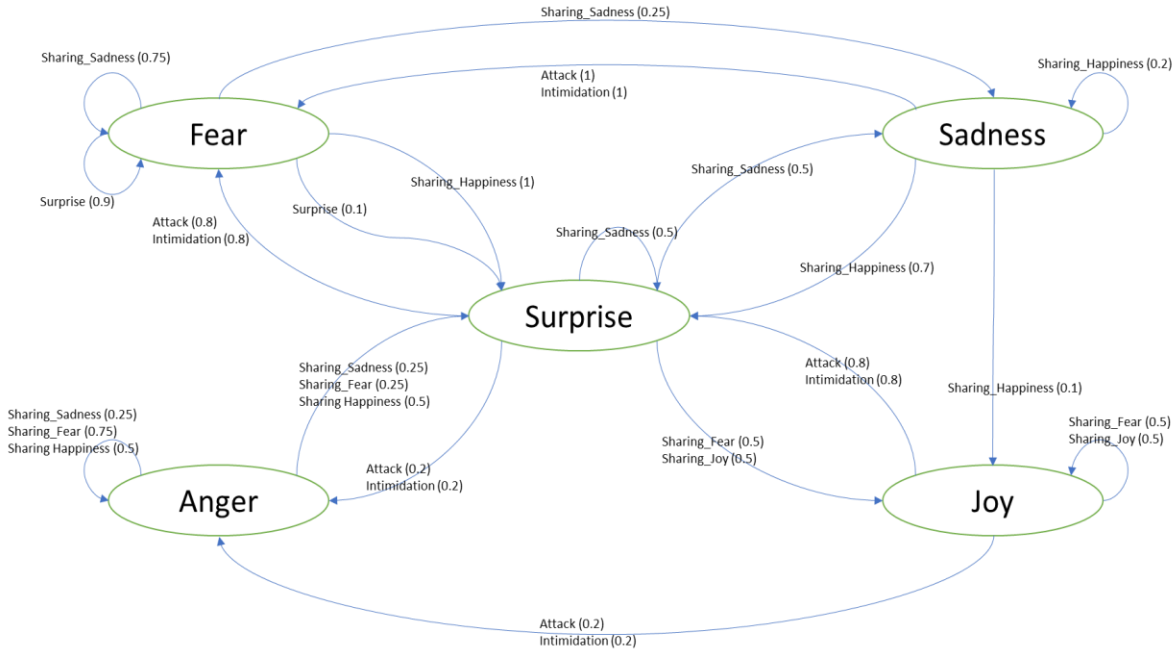

Figure 17: State-transition Diagram for "Architect (INTJ)" Personality

Another example is the state-transition diagram for the "Advocate (INFJ)" personality, which is shown in the following figure:

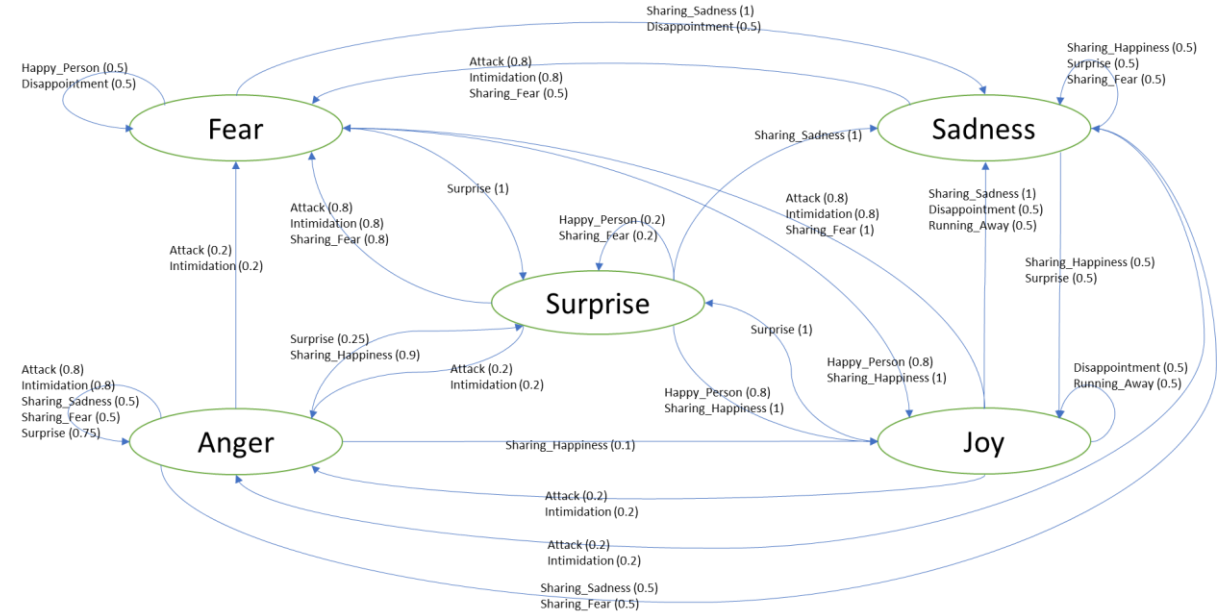

Figure 18: State-transition Diagram for "Advocate (INFJ)" Personality

A third example is the state-transition diagram for the "Commander (ENTJ)" personality, which is shown in the following figure:

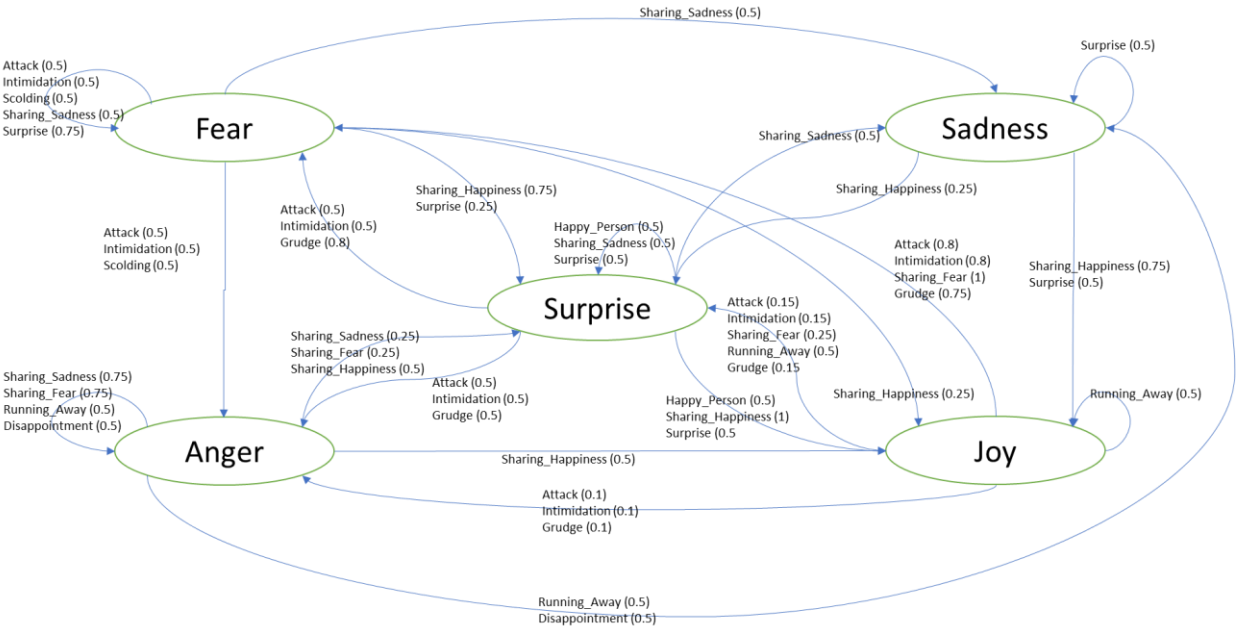

Figure 19: State-transition Diagram for "Commander (ENTJ)" Personality

Please note that for any scenic action which is not shown associated to a state transition, it is implicit that the emotional state remains unchanged.

Each state-transition diagram associated to a personality is implemented by means of a JSON file; as examples, the JSON files representing the "Architect", "Advocate" and "Commander" personalities are provided respectively in appendixes A.3, A.4 and A.5.

3.4 Execution of Emotional Reactions

Once a new emotional state has been reached (as a consequences of actor's scenic action and of the robot's "personality), the robot has to express it by moving around, changing its posture or bending his "eyes". In this way an interaction with the human actor can be achieved in a realistic way.

In our context, "reactions" are defined as the complex of movements performed by the robot after it has interacted with a human actor, with the aim to communicate feelings and emotions in a way which can be recognized by a human observer.

Reactions has been assumed to be essentially "specular" to the "scenic actions" defined previously for the human actor:

1. Attack
2. Scolding
3. Intimidation
4. Grudge
5. Sharing Happiness
6. Happy person
7. Satisfaction
8. Sharing Fear
9. Running away
10. Sharing Sadness
11. Disappointment
12. Surprise
13. Disbelief (negative)
14. Astonishment
15. End Scene

The reactions generated by each couple scenic action - emotional state are defined in the following table:

Emotional State > Action	Joy	Sadness	Fear	Anger	Surprise
attack	happy_robot	disappointment	running_away	attack	disbelief
scolding	sharing_happiness	sharing_sadness	running_away	intimidation	astonishment
intimidation	satisfaction	sharing_sadness	running_away	attack	disbelief
grudge	sharing_happiness	sharing_sadness	sharing_fear	grudge	disbelief
sharing_happiness	sharing_happiness	disappointment	running_away	grudge	surprise
happy_person	sharing_happiness	sharing_sadness	sharing_fear	scolding	surprise
satisfaction	satisfaction	disappointment	sharing_fear	grudge	astonishment
sharing_fear	sharing_happiness	sharing_sadness	sharing_fear	scolding	surprise
running_away	sharing_happiness	disappointment	sharing_fear	attack	astonishment
sharing_sadness	happy_robot	sharing_sadness	sharing_fear	scolding	surprise
disappointment	happy_robot	disappointment	sharing_fear	scolding	astonishment
surprise	sharing_happiness	sharing_sadness	running_away	attack	astonishment
disbelief	happy_robot	disappointment	running_away	scolding	astonishment
astonishment	sharing_happiness	sharing_sadness	running_away	intimidation	astonishment
leaving_scene	end_scene	end_scene	end_scene	end_scene	end_scene

Table 1: Robot's Reactions as a function of emotional state and scenic action

3.4.1 Reaction's Characteristics

Each one of the robot' reactions has been defined in terms of the following "observables":

- Emotion expressed by the robot (by means of eyes and body movements or sound): Anger, Joy, Fear, Sadness, Surprise
- Direction of robot's movement: Forward (i.e. toward the actor) or Backward
- Speed of robot's movement: Fast, Slow, or no movement
- Proximity of actor and robot: Intimate, Not Intimate, Neutral

The robot's reactions are substantially specular to the actor's scenic actions (defined in section 3.2) and are implemented as follows:

Reaction	Emotion	Direction of Movement	Speed of Movement	Proximity
Attack	Anger	Forward	Fast	Intimate
Scolding	Anger	Forward	Slow	Neutral
Intimidation	Anger	Forward	Slow	Intimate
Grudge	Anger	Backward	Slow	Not Intimate + Neutral
Sharing Happiness	Joy	Forward	Fast	Intimate + Neutral
Happy person	Joy	N/A	Fast	Not Intimate
Satisfaction	Joy	Backward	Slow	Not Intimate + Neutral
Sharing Fear	Fear	Forward	Slow	Intimate + Neutral
Running away	Fear	Backward	Fast	Not Intimate + Neutral
Sharing Sadness	Sadness	Forward	Slow	Intimate + Neutral
Disappointment	Sadness	Backward	Slow	Not Intimate + Neutral
Surprise	Surprise	Forward	(no move)	Intimate + Neutral
Disbelief (negative)	Surprise	Backward	Slow	Not Intimate + Neutral
Astonishment	Surprise	(no movement)	(no move)	Neutral + Not Intimate

Table 2: Implementation of Reactions

When the "end_scene" reaction has to be performed, the robot will simply stay in its current emotional state while its controller SW will exit from the part dealing with the interactive session and go back to the execution of a predefined Script (see section 5.4.3).

3.4.2 Expression of Emotions

To express its emotional state our robot can only resort to its movements; we now list hereafter the movements that the robot can make depending on its emotion.

"Base Linear" and "Base Angular" indicate the linear and angular movement of the base. For each cell, the first row indicates the direction of movement, the second the speed.

Emotion	Eyes	Body	Base Linear	Base Angular
Anger	Up-down *Fast*	Bow back *Fast*	(see "Reactions" table)	Rotate *Fast*
Sadness	Down *Slow*	Bow *Slow*	(see "Reactions" table)	Rotate *Slow*
Fear	Down-cross *Fast*	Bow *Fast*	(see "Reactions" table)	Rotate *Fast*
Happiness	Up *Fast*	Right-left *Fast*	(see "Reactions" table)	Rotate *Medium*
Surprise	Divide *Fast*	Bow back *Fast*	(see "Reactions" table)	None *None*

Table 3: Possible emotional movements

Below is an explanation of the meaning of each movement:

- Anger:
  Anger is a strong feeling, so the movements must be fast. The robot must therefore appear agitated but at the same time strong.
- Sadness:
  Sadness is a feeling that indicates weakness, so the robot must show itself as such by trying to hide, closing its body, and possibly simulating crying. The movements are slow.
- Fear:
  Fear also indicates weakness, but the situation has coincided, the movements therefore indicate a situation in which you want to hide and you don't know what to do. The speed is fast.
- Happiness:
  Happiness expresses an agitated situation, so the movements are fast. The robot rejoices by moving its eyes up and down and rotating on itself.
- Surprise:
  The surprise is an unexpected situation, the speed is fast, the robot divides its eyes making a movement as absurd as the situation in which it finds itself.

4. Hardware Architecture

The hardware of the robot has been completely reused from previous projects, in particular from the “Robocchio” platform developed in AIRlab and its complements developed in the frame of the graduation thesis of L. Bonetti [28]. In the following sections a brief description of this architecture is provided.

4.1 Platform: TriskarOne

The core of the hardware is the already developed robotic platform called “TriskarOne", a 3-wheeled omnidirectional robot built at AIRLab years ago and already used for other projects. Its scope was to play a game called “Robotower" in which it had to crash into plastic towers while the human player was trying to protect them using his body [32].

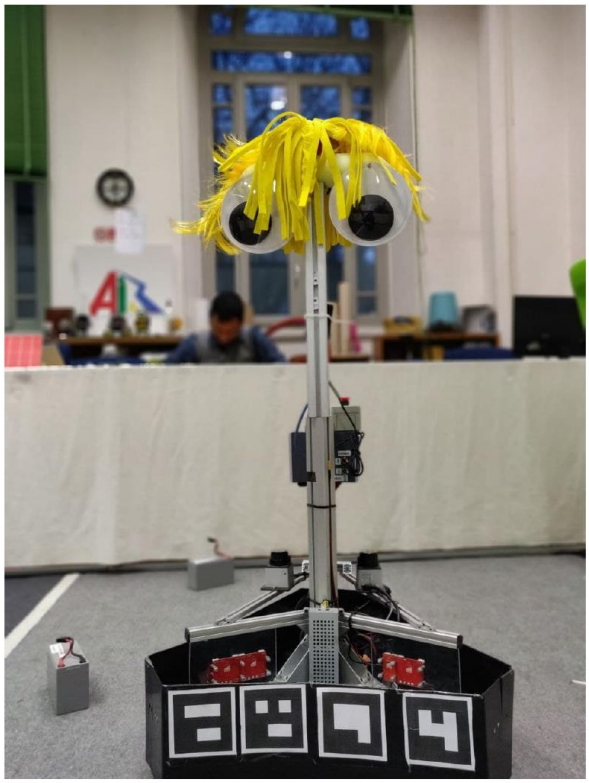

Figure 20: TriskarOne platform before application of the body

The robot has a triangular base, with one wheel mounted on each vertex. The base and the electronics inside are protected by plastic bumpers. On the base, near the motors, are collocated 2 batteries and the power unit, the motor drivers and 2 laser scanners. At the center of the base is attached a vertical shaft to which an Intel i7 computer is secured, together with a box containing an Arduino board, a box containing the switches to turn it on and an emergency button.

An external body has been applied to the basic platform during the development of the "Robocchio" project. The waist was designed mimicking the kinematic proposed by the much smaller robot Blossom [60].

Moreover, two eyes have been mounted, each one composed by a polystyrene sphere which represents the eyeball, connected to a PVC elastic pipe equipped with 4 nylon wires which are rolled up or released on by 2 servo motors mounted at the base of the eye; depending on the angle of rotation of each of the two motors the eye can rotate and look in different directions.

The final aspect of the robot is shown in the following figure:

Figure 21: The exterior aspect of the robot

4.2 Computer

The SW main controller node runs on a Shuttle DH310 Mini PC [8] which mounts Intel I7-8700 CPU, a DDR4 RAM and a fast 240 GB SSD. It has 8 USB-ports, 2 HDMI ports, 1 audio output port and 1 audio input port.

Figure 22: Shuttle DH310

4.3 Sensors and Actuators

4.3.1 Arduino

To control the servo-motors and the battery status we used Arduino Uno [9]. Arduino Uno is a microcontroller board based on the ATmega328P. It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz ceramic resonator, a USB connection, a power jack, an ICSP header and a reset button.

Figure 23: Arduino Uno

4.3.2 Battery

The robot's power supply is composed by 2 lead batteries each having a nominal voltage of 12 V and a capacity of 9Ah, thus providing a total power of 216Wh The charge state of the battery is checked using a voltage divider.

4.3.3 Eyes and Body

The eyes and the body are moved by servo motors because they are easier to control at a position than a DC motor, and because of their cost effectiveness. In particular MG996R are used because they are light, small, and strong enough. They are supplied at 6V and provide 11 kgf·cm of torque. The maximum operating current is 900 mA. On each motor there are wheels that pull and let go of the wires allowing the movement.

Figure 24: MG996R

4.3.4 Laser Scanners

The robot mounts 2 Hokyuo URG-04LX-UG01 laser scanners. The light source of the sensor is an infrared laser. The scan area is 240º semicircle with maximum radius 4000mm. Pitch angle is 0.36º and each sensor outputs the distance measured at each point. Distance measurement is based on calculation of the phase difference between emitted and detected signals, due to which it is possible to obtain stable measurement with minimum influence from object's colour and reflectance. The sensor is designed for indoor use only. Lasers can have several sources of error. As they rely on light reaction, a surface that does not reflect enough light (such as a very dark surface) would cause the sensor to think there is no obstacle in front of it. The same effect could have a surface that allows light to penetrate, such as a glass, but also a fabric with very large textures. Furthermore, walls that have irregularities (such as electrical sockets) lead the

rays to not reflect correctly, but to go in unexpected directions. Finally, it is possible that completely causal and unexpected measurements may occur due to dust or the passage of small objects. It is therefore good to take these kinds of errors into account when developing algorithms that are based on the measurements of these laser scanners.

Figure 25: Hokuyo URG-04LX-UG01

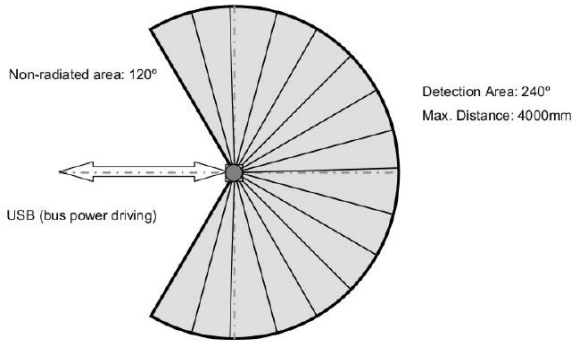

Figure 26: Hokuyo's range

4.3.5 Motorization

4.3.5.1 Hardware

Holonomic platforms are characterized by the possibility to move in any direction without the necessity to have a specific orientation, i.e., they are free to move taking any desired orientation. This type of movement requires a specific kind of wheel, as the one that is shown in the figure below. This type of wheels can lead to some sources of uncertainty. In fact, due to their ability to slide in any direction, the possibility of errors in the calculation of the inverse kinematics due to small drift increases. This type of error must be taken into account when calculating odometry.

Figure 27: Holonomic Wheel

This kind of platforms require at least three motors and wheels to move. The figure below depicts a holonomic platform with three motors. In this configuration each motor is placed every $2\pi/3$ angle on a circular reference and each wheel has a rotation of $\pi/2$ w.r.t. the motor's axis

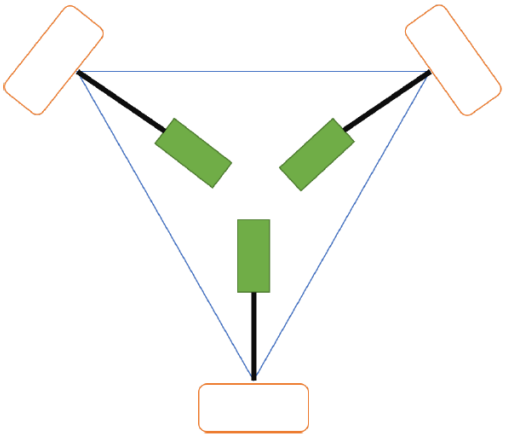

Figure 28: Motors configuration

4.3.5.2 Motors

The three motors are MAXON 118798 DC motor RE36 GB 70W KL 2WE, whose characteristics are reported in the figure below. On each motor a 110513 tacho ENCODER HEDS 5540 500IMP 3K is mounted to get the speed of the motor. Encoders contain a single Light Emitting Diode (LED) as its light source. The light is collimated into a parallel beam by means of a single lens located directly over the LED. Opposite the emitter is the integrated detector circuit. This IC consists of multiple sets of photodetectors and the signal processing circuitry necessary to produce the digital waveforms. The code-wheel rotates between the emitter and detector, causing the light

beam to be interrupted by the pattern of spaces and bars on the code-wheel. The photodiodes detect these interruptions and send signals to the signal processing circuitry that produces the final outputs that is an index pulse PO which is generated once for each full rotation of the code-wheel.

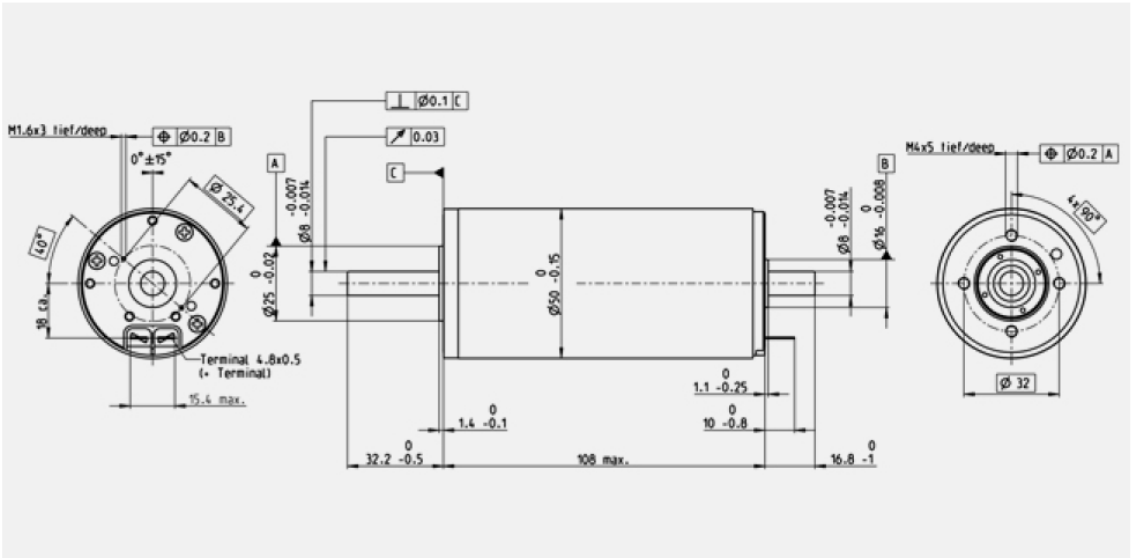

Figure 29: MAXON 118798 DC motor characteristics

4.3.5.3 Drivers

The low-level motors actuation and their interface between the ROS system have been realized with the Nova Core modules based on STM32-chip. The provided modules allow to control different type of motors that can be modelled as a second order system, where the input is the voltage applied to the motor armature and output variable is the motor angular speed.

4.3.5.4 Kinematics

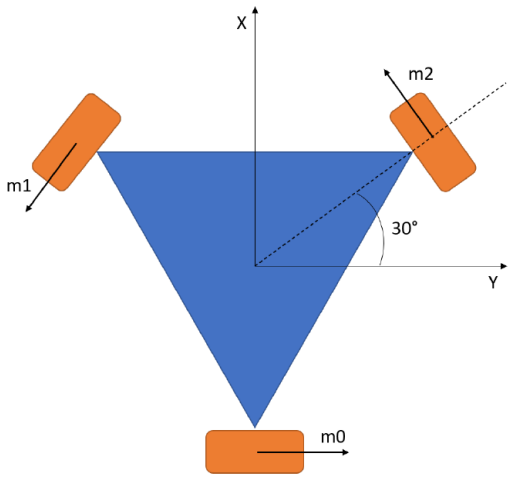

Figure 30: Holonomic platform with three motors in a Cartesian plan

(m1, m2 and m3 represent the motors)

Using the configuration shown in the figure below, it is possible determine the velocity of each motor < m_1;m_2;m_3 > given the desired velocity triplet < V_x; V_y; $V_!$ >, where V_x and V_y are the velocities in the x and y axis directions, respectively, and ! is the rotational velocity with respect to the center of the robot. Then, the system could be described as follows:

$$\begin{bmatrix} V_x \\ V_y \\ \omega \end{bmatrix} = \begin{bmatrix} 0 & -\frac{1}{\sqrt{3}}R & \frac{1}{\sqrt{3}}R \\ -\frac{2}{3}R & \frac{1}{3}R & -\frac{1}{3}R \\ \frac{R}{3L} & \frac{R}{3L} & \frac{R}{3L} \end{bmatrix} \begin{bmatrix} m_1 \\ m_2 \\ m3 \end{bmatrix}$$

where R is the wheel's radius, L is the distance between the center of the configuration and a wheel. The contribution of each motor to each velocity component can be described as follows:

4.3.5.5 Speed Modulation

The velocity of each motor is given by this formula:

$$V = \frac{2 * \pi * NP}{PT * \Delta t} \frac{rad}{s}$$

where NP is the quantity of pulses counted during the time window and PT is the quantity of pulses that are necessary to do a whole turn in the wheel, Delta_t is the time elapsed between two consecutive calculations.

4.3.6 Network Communication

To allow communication between the robot and the operator's computer, a D-Link GO-DSL-N150 router was used which generates a network to which both devices are connected.

4.3.7 Power Supply

The DC motors are powered at 24V through the drivers. The computer is powered at 19V through a DCDC-USB-200. Arduino and the servomotors are powered at 6V through two DC-DC LM2596, each capable of providing up to 3A (it was necessary to use two DC-DCs due to the large current consumption by the servos when they are particularly stressed).

Figure 31: DC-DC LM2596

5. Software Architecture

The software consists of a set of concurrent modules developed within "Robot Operating System" (ROS) framework, as describe in the following sections.

5.1 Overall Concept View

A simplified representation of the overall SW architecture is presented in the following conceptual diagram:

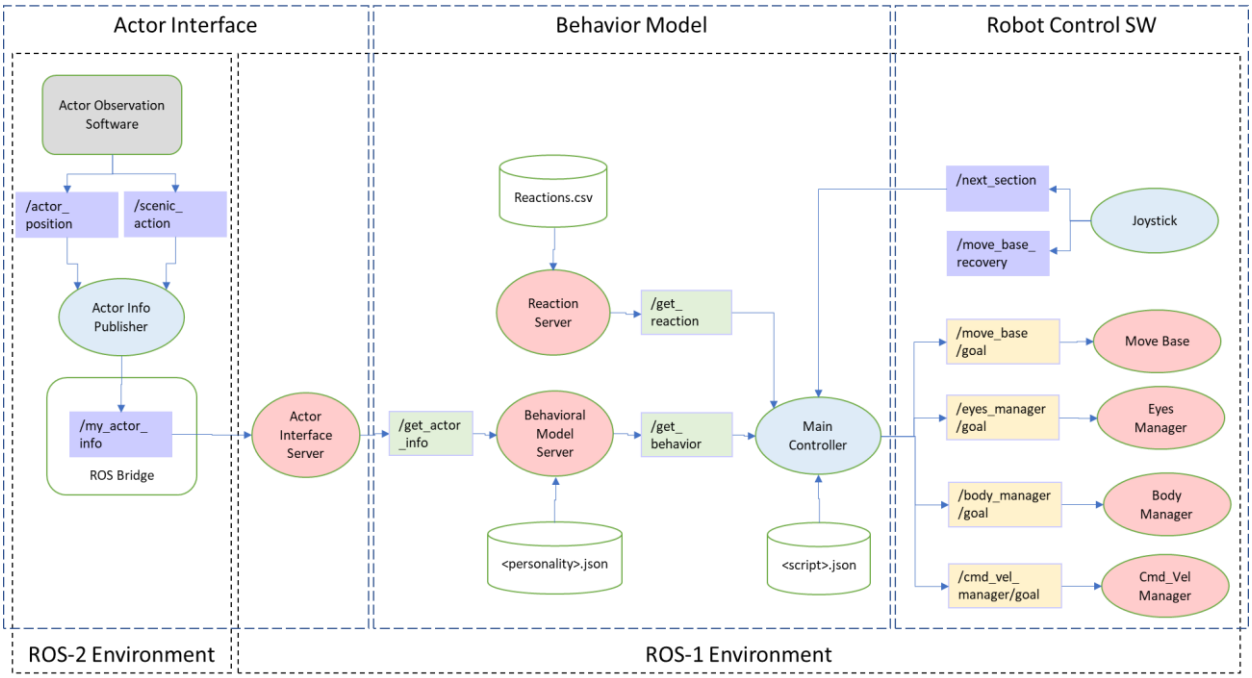

Figure 32: Overall Concept View

With reference to figure above (from left to right), the software architecture consists essentially of the following main parts:

1) The SW in charge of observing the performance of the human actor.
   As stated in section 1.1, this SW package is the object of a dedicate graduation thesis currently under development at AIRlab.

2) A node in charge of acquiring the information on the actor's position and scenic actions and transmit it through the ROS Bridge.
3) A ROS-2 standard library ("ROS Bridge") which allow inter-process communications between nodes running in ROS-1 environment and nodes running in ROS-2 environment.
4) A node in charge of making the information on the actor's position and scenic actions available to the other nodes of the architecture.
5) A node in charge of implementing the robot's Behavioural Model, according to the strategy defined in chapter 3.3.1 above.
6) A node in charge of defining the robot's reaction as a function of the recognized scenic actions and of its current emotional state
7) A "Main Controller" node which allows to:
   - execute predefined "scripts", i.e. list of actions to be executed in sequence by the robot upon the occurrence of specific trigger conditions (see section 5.4.3.4)
   - start an "interactive" session with the human actor, therefore passing from executing predefined actions to "improvising" its behaviour
   - acquire from the Behavioural Model the information concerning the actor's scenic actions and the robot's emotional state
   - executing the robot's reaction by giving appropriate commands to the modules in charge of controlling the robot's navigation and its body and eyes movements
8) A set of nodes dedicated to control the robot's navigation around the stage and to control its body and eyes movements. This software has been developed in the frame of previous projects carried on at AIRlab, in particular for the graduation thesis of L. Bonetti [28] and has been as far as possible reused "as is".

5.2 Robot Operating System (ROS) Framework

5.2.1 Generalities

ROS (Robot Operating System) is an open-source, meta-operating system that provides the services typical of an operating system, including hardware abstraction, low-level device control, implementation of commonly used functionality, message-passing among processes, and package management. It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers. The ROS

runtime "graph" is a peer-to-peer network of processes (potentially distributed across different machines) that are loosely coupled using the ROS communication infrastructure. ROS implements several different styles of communication, including synchronous RPC-style communication over services, asynchronous streaming of data over topics, and application-wide parameters management services.

ROS is one of the most used frameworks in robotics. It has a widespread support and there are many libraries already built for it. It easily supports multitasking and synchronization between multiple machines. It can be programmed both in Python and C++ so it is very easy to integrate with systems written in these languages.

ROS has been developed since 2007. Many versions have been released; the more recent one is ROS Noetic [5], which runs on Linux Ubuntu 20.04.

Starting from 2014, a new system named ROS-2 has been developed by the ROS community. ROS-2 shares many operational concepts with the original ROS environment ("ROS-1") but introduces also seeral different characteristics both at syntactical and architectural level, therefore a program written under ROS-1 cannot be executed in ROS-2. However, a dedicated package named "ROS Bridge" allows programs running in ROS-1 to communicate with programs running in ROS-2 and vice-versa.

In the following, the term "ROS" will be used when dealing with concepts and characteristics present both in ROS-1 and ROS-2, while "ROS-1" or "ROS-2" will indicate features proper of only one of the two environments.

5.2.2 Structure of a Project in ROS

A project in the ROS environment includes:

- Nodes:
  Nodes are processes that perform computation and control tasks. ROS is designed to be modular at a fine-grained scale; a robot control system usually includes many nodes. For example, one node may control a laser range-finder, one node may control the wheel motors, one node may perform localization, one node may perform path planning, one node may provide a graphical view of the system, and so on.
- Packages:
  Packages are the main unit for organizing software in ROS. A package may contain ROS runtime processes (nodes), a ROS-dependent library, datasets, configuration files, or anything else that is useful to organized together. Packages are the most atomic build item and release item in ROS, meaning that the most granular bject that one can build and release is a package.

- Workspaces:
  All the packages of a given project are grouped in a Workspace, where the "build" operations can be performed using dedicated commands:
  - "catkin_make" in ROS-1 environment
  - "colcon_build" in ROS-2 environment

  By means of these commands it is possible to build the complete workspace or only a specified subset of the packages included in the workspace.

5.2.3 Inter-Process Communications Mechanisms and Data Structures

A ROS system can be represented as a set of nodes in a graph structure, connected by edges which symbolize the communications between them. Three basic communication mechanisms are possible:

- **Topics**: Messages are routed via a transport system with publish/subscribe semantics. A node sends out a message by publishing it to a given topic. The topic is a name that is used to identify the content of the message. A node that is interested in a certain kind of data will subscribe to the appropriate topic. There may be multiple concurrent publishers and subscribers for a single topic, and a single node may publish and/or subscribe to multiple topics. In general, publishers and subscribers are not aware of each other existence. The idea is to decouple the production of information from its consumption. Logically, one can think of a topic as a strongly typed message bus. Each bus has a name, and anyone can connect to the bus to send or receive messages as long as they are of the right type.

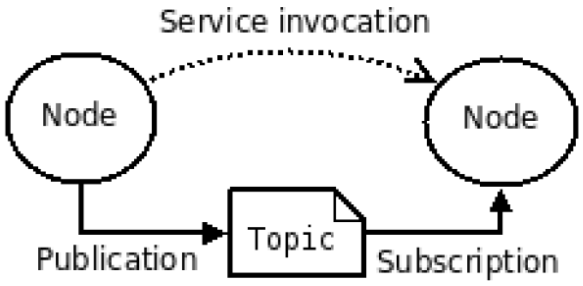

Figure 33: Communication between nodes using topics

- **Services**: The publish/subscribe model is a very flexible communication paradigm, but its many-to-many, one-way transport is not appropriate for request/response interactions, which are often required in a distributed system.

Request/reply is done via Services, which are defined by a pair of message structures: one for the request and one for the reply. A providing node offers a service under a name and a client uses the service by sending the request message and waiting for the reply. ROS client libraries generally present this interaction to the programmer as if it were a remote procedure call.

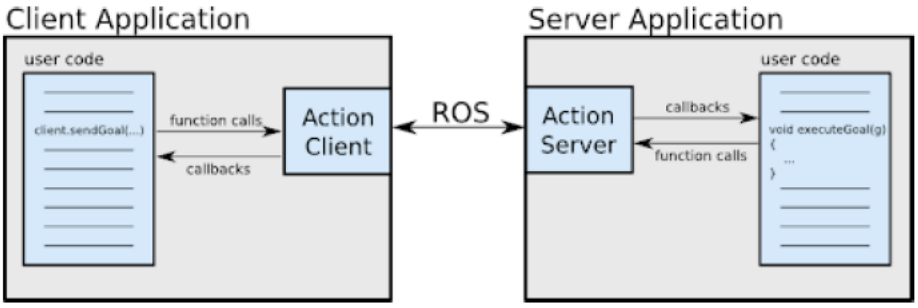

Figure 34: Client-Server Communication in ROS

- **ActionLib**: In some cases, if the service takes a long time to execute, the user might want the ability to cancel the request during execution or get periodic feedback about how the request is progressing. The ActionLib package [7] provides tools to create servers that execute long-running goals that can be pre-empted. It also provides a client interface in order to send requests to the server.

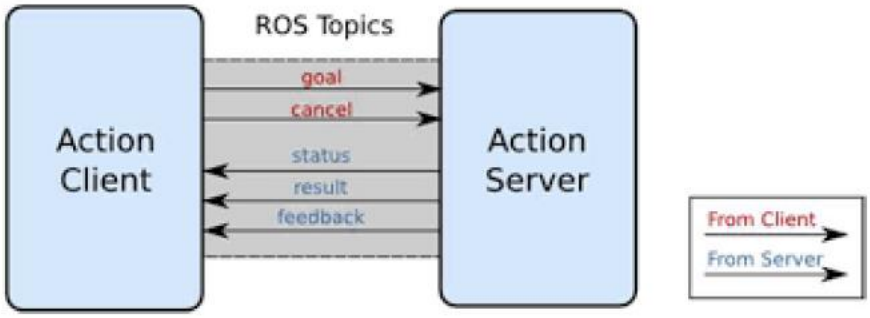

Figure 35: ActionLib interface

For each one of the mechanisms listed above, a specific data structure is foreseen:

- Message (msg) types: Message descriptions, stored in <my-package>/msg/MyMessageType.msg, define the data structures for messages exchanged using topics.

- Service (srv) types: Service descriptions, stored in <my-package>/srv/MyServiceType.srv, define the request and response data structures for Services in ROS.
- Action types: Action descriptions, which includes Goal, Feedback and Result, define the possible robot actions used by ActionLib [7].

5.2.4 ROS-1 vs. ROS-2

5.2.4.1 ROS-1 Use Case

Since its appearance, ROS has rapidly become the standard environment for the development robotic applications. However, it was conceived having in mind a typical "academic" use case, which usually presents the following characteristics:

- a single robot
- workstation-class computational resources on board
- no strict real-time requirements
- excellent network connectivity
- maximum flexibility, with nothing prescribed or proscribed

When moving toward industrial applications, it rapidly became very difficult to cope with these constraints. Instead of heavily rewriting the existing ROS kernel (which could have created problems to the ecosystem of applications using it), the ROS community decided to start the development of a new middleware suite named ROS-2 and to foresee a rather long period during which the 2 systems could coexist.

5.2.4.2 Evolution toward ROS-2

ROS-2 has been developed since 2015 in order to overcome the ROS-1 limitations [4] and satisfy the need for the following use cases:

- **Multi-robot system**: Multi-robot system is a key issue in the field of robotics. It can solve the problems of insufficient performance and unavailability of a single robot. However, there is no standard method for building a multi-robot system in ROS1.
- **Multi-platform**: ROS-1 is based on Linux and cannot be applied or has limited functions on Windows, MacOS, RTOS and other systems. This places higher requirements on robot developers and development tools, and also has great limitations.

- **Real-time**: Robots in many application scenarios have high requirements for real-time performance, especially in the industrial field. The system needs to achieve hard real-time performance indicators, but ROS-1 lacks real-time design, so it is stretched in many applications.
- **Network connection**: The distributed mechanism of ROS-1 requires a good network environment to ensure data integrity, and the network does not have data encryption, security protection and other functions. Any host in the network can obtain the message data issued or received by the node.
- **Production**: The presence in ROS1 of a "Master Node" introduces a decisive dependence on a single-point failure and therefore a lack of robustness, which makes many robots transition from research to market very difficult.

The differences between ROS1 and ROS-2 frameworks [15] are summarized in the following table:

	ROS1	ROS2
OS support	Linux	Linux, Windows, MacOS
ROS Master	Yes	No, distributed system
Client libraries	rospy, roscpp	rcl → rclpy, rclcpp
Language support	Python 2, up to C++ 11	Python 3, C++ 14
Way to structure your nodes	No special structure	OOP

Table 4: Differences between ROS-1 and ROS-2

A simplified block diagram comparing ROS1 and ROS-2 architectures [16][19][20] is shown in the following figure:

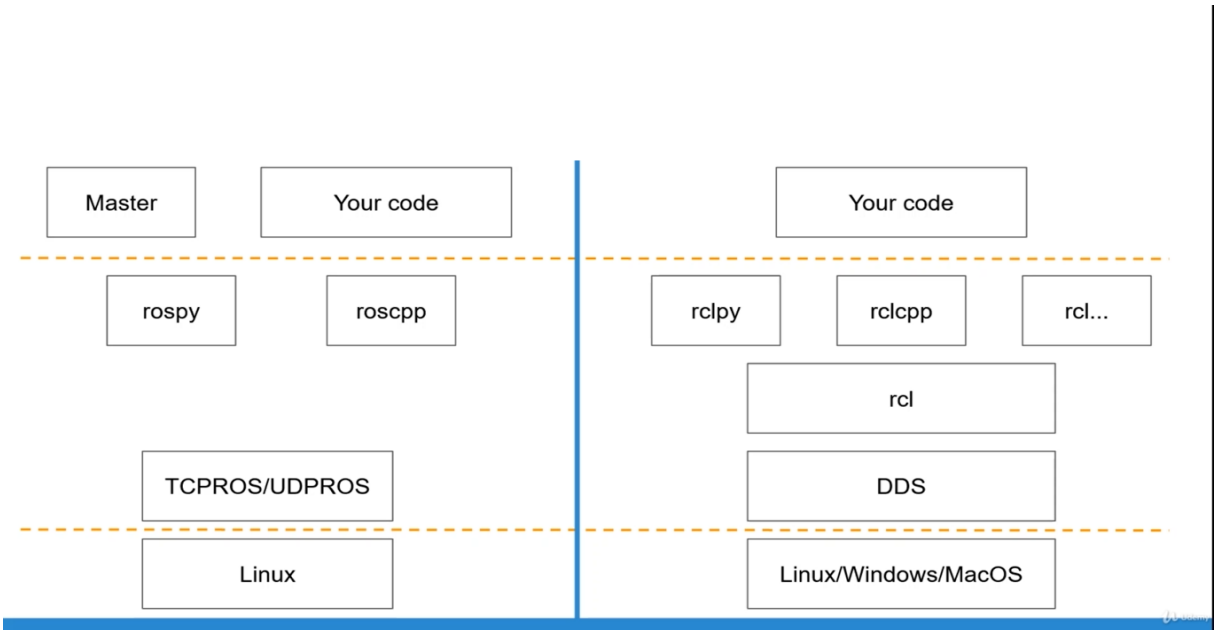

Figure 36: Comparison between ROS1 and ROS-2 architectures

5.2.4.3 Selecting ROS Version

When starting the implementation of our project, we had to perform a trade-off about the version of ROS to use. Our choice has been based on the following considerations:

1. Our task is 100% covered by the typical ROS-1 use case described in section 5.2.4.1 above.
2. However, the global trend (both in academic and industrial contexts) is currently to move toward the ROS-2 framework.
3. Nevertheless, the support to ROS-1 by the worldwide ROS development community is anyway ensured up to the end of 2025.
4. As explained in the Introduction, our work performs a very significant reuse of SW already developed in the frame of previous projects; in particular, all the SW in charge of controlling the robot's movements and navigation is reused from the "Robocchio" project completed in 2021 which was written entirely under ROS-1.

The last point was the decisive one. Reusing the available SW under ROS-2 would not have required a pure "porting" activity but an almost complete re-development because:

- the differences between ROS-1 and ROS-2 are not only at a pure syntactical level but include a conceptually different approach;

- the inter-process communication mechanisms, even if functionally similar (both ROS-1 and ROS-2 use topics, services and ActionLibs), require to be defined and used in different ways;
- the SW in charge of controlling to robot's HW platform is made up by several concurrent nodes with a considerable number of interfaces (topics, services, actions etc.), so every modification has a set of side effects which should be analysed carefully and could easily introduce errors very difficult to debug.

Therefore we decided to develop our SW in the ROS-1 environment, in order to be able to concentrate our efforts on the main objective: model the behaviour of the robot in a reactive and improvisational context.

However, we also decided to "open" our SW to the communication with other packages developed under ROS-2. This is possible using a standard library package named "ROS Bridge" which executes under ROS-2 and is able to connect topics and services between the ROS-2 environment. In this way it has been possible to interface our SW with the scenic action recognition SW which, making no significant reuse of legacy SW, has the possibility to be fully developed under ROS-2.

5.2.5 ROS Graphs

In the following sub-sections is reported the information about each component in the ROS-based code. The meaning of the figures is understandable thanks to the legend reported here below.

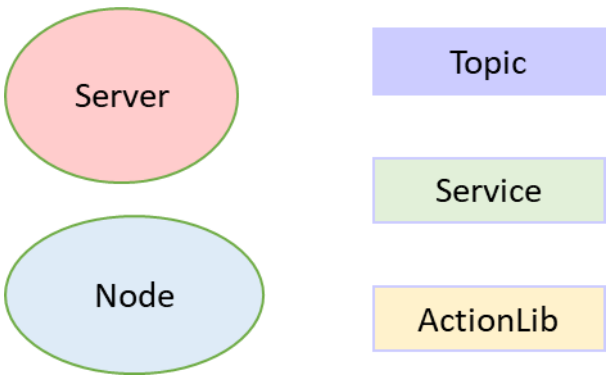

Figure 37: Legend for the figures in this chapter

5.3 Actor Interface SW

5.3.1 Acquisition of Actor's Scenic Actions

As stated in section 1.2, the SW in charge of recognizing the scenic actions performed by the actor is the object of a dedicate graduation thesis currently under development at AIRlab. We therefore see this SW package as an external interface.

The interface with scenic action recognition SW is implemented by means of a ROS topic. The recognition SW act as "Publisher": it sends on this topic the information concerning the actor' position and the scenic action which is acquired and processed by the "Actor Interface" node of our SW (see below) which act as "Subscriber".

The position of the actor is defined as a couple of spatial coordinates, while the scenic action is assume to be one in the list defined in section 3.2 above.

The scenic action can be one in a list of possibilities defined together with our colleague (see section 3.2 above). However, our SW implementing the robot's emotional life and reactions is independent from this list: in case a new or different scenic action is defined, it would be sufficient to update the JSON files defining the various personalities to implement the new behaviour without modifying a single line of code.

5.3.2 Actor Info Publisher

The Actor Observation SW provides a real-time state of the actor by continuously publishing messages over the /actor_position topic; these messages are composed of the following: a discrete categorization of the position ("front", "back", "left", "right") and a numerical distance ("coordX", "coordY") of the actor w.r.t. the robot frame.

Moreover, the topic /scenic_actions contains a single action value from the available pool ("attack", "scolding" etc. as defined in section 3.2), representing the last act of the actor. Due to the different synchronism between the information published on the two topics (i.e. the actor position is available much more frequently and almost in real-time, while the scenic action identification which requires more time to be computed) we decided to implement a mechanism to group all the information coming from the two topics and re-publish them together.

This approach allows to overcome another important implementation issue: while the actor observation SW runs in the ROS-2 framework, the robot reaction module runs in the ROS-1 framework because it needs to include the existing legacy implementation. An interface between the two environments is needed. For this reason we implemented this ROS-2 node whose task is to acquire the messages coming from the actor

observation SW and funnel them through the ROS Bridge[6] in order to make them available for the robot behaviour and reaction modules.

5.3.3 ROS-1 to ROS-2 Bridge

The ROS-1 to ROS-2 Bridge [6] is a package running in the ROS-2 environment providing a network bridge which enables the exchange of messages between ROS 1 and ROS 2 workspaces.

We used it to connect our Actor Interface node (acting as Subscriber of the /my_actor_info Topic, see section 5.3.4) to the external SW in charge of recognizing the actor movements and scenic actions, which act as Publisher of this information on the same topic.

The standard ROS Bridge package is able to automatically connect only topics having the same name and exchanging only messages with "standard" types (e.g. strings and integers). In principle it is possible to exchange messages with custom-defined types but this feature would require to regenerate the Bridge package from source, so for the sake of simplicity we decided to link topics having the same name and exchanging messages of type "String", where actually the string is a complete JSON text file where we are able to specify all the needed information.

5.3.4 Actor Interface Server

The Actor Interface server is the ROS node in charge to interface the external software dedicated to the observation of the human actor.

When started, this node initializes all its data structures and all the inter-process communication mechanisms it will need during execution:

- It subscribes to the /my_actor_info topic to get the information from the actor observation SW when it sends it, then is stores it inside an internal data structure
- It starts the /get_actor_info Service to provide the information about the actor to the Behavior Model server when it requests it

This process allows to de-couple the flow of information from the actor observation SW from the execution of the Behaviour Model.

5.4 Behaviour Model

5.4.1 Behavioural Model Server

As explained in section 3.3.3, the behaviour of the robot is modelled by means of a non-deterministic finite-state automaton (NFA)

The state-transition function defining each personality is defined by means of a specific table contained in a .JSON file, which can be filled and modified using a standard text editor. This file is then used by the Behaviour Model node to actuate the evolution of the emotional state.

When started, this node initializes all its data structures and all the inter-process communication mechanisms it will need during execution:

- It reads the robot's personality and initial emotional state from dedicated ROS parameters
- It loads the JSON file for the NFA corresponding to the selected personality into an internal table structure
- It loads the CSV file specifying the robot's reaction for each scenic action and current emotional state into an internal table structure (at present this table is fixed but it could be made dependent on the personality, see section 7.2.5.1)
- It connects to the /get_actor_info Service provided by the Actor Interface Server
- It starts the /get_behavior Service to provide the information concerning the actor (scenic action, position, orientation), the robot's emotional state and the robot's reaction to the Main Controller when it requests it

After initialization, this node enters a loop waiting for service requests on the /get_behavior Service. When it receives the request:

- ➢ it gets the information concerning the actor from the /get_actor_info Service
- ➢ it calculates the next robot's emotional state on the basis of the state-transition table associated to the defined personality
- ➢ it calculates the robot's reaction using the Reactions table
- ➢ it provides all the received and calculated information to the Main Controller

5.4.2 Reaction Server

The Reaction Server is the node in charge to identify the list of steps that the robot have to execute in order to perform the reaction communicated by the Behaviour Model server (via the Main Controller).

When started, it it immediately creates the /get_reaction Service, then enters a loop waiting for service requests. When it receives the request, it elaborates the list of steps to be executed to perform the specified reaction. The reaction steps can be:

- "atomic actions", i.e. the same actions that can be specified in a Script (move_base, move_body, move_eyes etc.)
- "complex actions", i.e. a combination of atomic actions to perform a more "sophisticate" behaviour (e.g. face the actor, approach him, run away, etc.)

It then passes the list of reaction steps to the Main Controller for execution.

5.4.3 Main Controller

The main controller is the node in charge of reading the script, deciding when to start a section, calling all the nodes in charge of performing the selected actions and checking if all the actions have been done. It is also in charge to manage the interactive session with the human actor by interfacing with the Actor Interface, Behavior Model and Reactions servers.

5.4.3.1 Initialization Phase

When started, the Main Controller node initializes all its data structures and all the inter-process communication mechanisms it will need during execution.

It subscribes to topics:

- /next_section
- /move base recovery

It behaves as a Client for the following ActionLibs:

- /move_base
- /eyes_manager
- /body_manager
- /cmd_vel_manager

It behaves as a Client for the following Services:

- /get_behavior
- /get_reaction

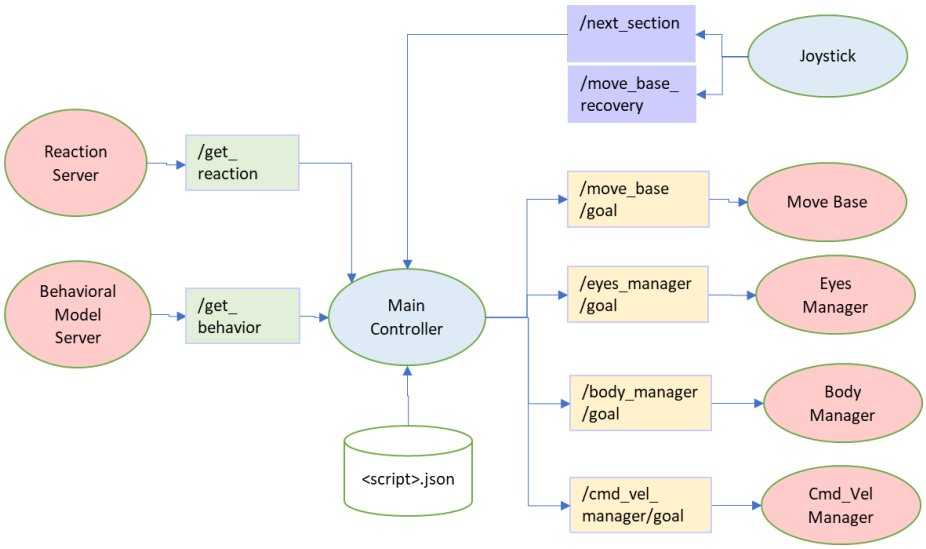

Figure 38: Main Controller and published/subscribed topics

5.4.3.2 Basic Action Routines

The control of robot's movements is performed by means of a set of "basic action routines" which manage the interfaces with the nodes in charge of controlling the robot's actuators (see section 5.5):

- **move_base()** sends a Goal to the *move_base* package via the related ActionLib
- **move_eyes()** sends a Goal to the *move_eyes* package via the related ActionLib
- **move_body()** sends a Goal to the *move_body* package via the related ActionLib
- **manual_move()** sends a Goal to the *cmd_vel_manager* package via the related ActionLib
- **check_action_status()** verifies that the execution of a previous move routine has been successful and if not it retries it until the action is aborted

These routines are called during the execution of a Script (see section 5.4.3.4) or during the performance of a scene interacting with the human actor (see section 5.4.3.5).

5.4.3.3 Main Loop

After initialization the Main Controller node runs a main loop aimed to the execution of predefined Scripts. In each iteration of the loop, the Main Controller performs the following actions:

- It extracts the current section and all the information saved in it from the script.
- It reads when it must start performing the indicated actions, if it is necessary to wait for someone to speak or for a command to be given, it puts itself on hold (waiting for the nodes in charge of verifying this information).
- When the actions can be executed, the Main Controller calls all the nodes in charge of performing the requested actions passing the necessary information.
- Then it waits until all the nodes have notified it that they have finished their tasks, after which it moves on to the next section.
- It is also responsible for taking a pause without any action (if required).
- If "move_base" has been called, it is responsible for dealing with the case in which the robot was unable to reach the desired position on the map and asking the operator to decide whether to try again or to proceed to the next action.
- A very special action which can be specified in a Script is the starting of a "scene": this action signals to the robot to start an interactive session with the human actor, during which the loop scenic-action-recognition -> emotional-state-evolution -> reaction-execution is performed.

5.4.3.4 Script Execution

A script is a JSON file containing all the actions the robot must do. It is divided in sections, each one containing all the actions the robot must do at the same time.

Each section consists of 2 parts: trigger and actions. Trigger contains information about when the section should start, actions contains a list of all the actions the robot should perform. Every section is composed as follows:

{section#:

trigger: *trigger_type*

actions:{

move eyes: *move_eyes_data*

move body: *move_body_data*

move_base: *move_base_data*

manual_movement: *manual_movement_data*

start_scene: *start_scene_data*

do_nothing: *do_nothing_data*

end: *end_data* }

}

where:

- **trigger_type** is a string with values:
  - *after_precedent*: the section starts immediately after the precedent has finished
  - *after_command*: the section starts after a button on the joystick is pressed (the right arrow in our case)
- **move_eyes_data** is a list of a trio of integers, the first is the pause to make before performing the movement, the second is the type of movement of the eyes, the third is the speed. The type of movement can be:
  - Standard position
  - Look right
  - Look left
  - Look up
  - Look down
  - Cross eyes
  - Divide eyes
- The speed is a multiplier of a constant number, speed*constant is the time to wait for the eyes to move by one degree. The bigger is this number, the slower the eyes movement is.
- **Move_body_data** is a list of a trio of integers, the first is the pause to make before performing the movement, the second is the type of movement, the third is the speed. The type of movement can be:
  - Standard position
  - Bow front
  - Bow right
  - Bow left
  - Bow back

  The speed value is similar to the one related to the eyes.

- **Move_base_data** is a list that contains 4 float numbers, the first two are the abscissa and the ordinate representing the position the robot has to reach on the map, the second two represent the orientation.
- **Manual_movement_data** is a list that contains multiples of 7 float numbers. The first 3 numbers indicate the amount of space to move in < X, Y ; θ > direction, the second 3 numbers indicates the velocities in < Vx, Vy, Vθ > and the last number indicates the amount of time in seconds to wait between these movements.
- **Start_scene_data** is always a void mark (-1): these action indicates the start of an interactive session with the human actor
- **do_nothing_data** is a float indicating for how long the robot has to do nothing. This is the way to pause between sections
- **end_data** is always a void mark (-1): when actions contain the ”end” tag the script execution is finished.

Each action is executed calling the related Basic Action Routine and passing to it the data specified in the action line.

An example of a minimal Script is provided in Appendix A.2.

5.4.3.5 Scene Performance

When an action of the Script specifies the start of a scene, the Main Controller enters a dedicated loop when the interaction with the actor is managed:

- It gets from the /get_behavior Service provided by the Behavior Model server the information concerning the actor (scenic action, position, orientation), the robot’s emotional state and the robot’s reaction
- On the basis of the received information, it analyses the actor’s behaviour, then it passes the actor’s behaviour and the robot’s reaction to the Reaction server and gets back the list of reaction’s steps to be executed.
- It then proceeds to execute the specified reactions steps calling one or more Basic Action Routine (depending if a “complex” or “atomic” action have to be executed, see section 5.4.2).
- When the reaction to be executed is “end_scene”, the execution of the interactive session will be terminated. The Main Controller then exits from this loop and proceeds to the execution of the following section in the Script (see section 5.4.3.4).

5.4.3.6 Parameters

The following modifiable parameters can be used to pass information to the Main Controller:

- *script_path*: the path to the script we want to perform
  This parameter is used to change the script to be performed by the robot when the show is changed, but also between one scene and another if you prefer to divide the script into several scenes (in case you find it more comfortable).
- *section_to_start*: the section from which to start to read the script (default 1).
  This parameter is mainly used during rehearsals, to try out only some parts of the script without repeating it from the beginning, but also in the extreme case in which you have been forced to interrupt the execution at a point, from which you then want to restart.
- *personality*: the robot's personality.
  This parameter is used to select the appropriate NFA to be used for the evolution of the robot's emotional state (see section 3.3.4).
- *initial_state*: the emotional state that the robot enters at the start of the interactive session. It represent the "s0" state in the NFA (see section 3.3.3).
- *map_size*: the size of the simulated stage (assumed to be a square). It defines the limits for the movement of the robot and it is a safety parameter to avoid that the robot causes damages to itself or other objects by trespassing the border of the simulated stage.

Other parameters has been used during the test and debug phases but are not used during the robot's nominal operations

5.5 Robot Platform Control

As stated in section 5.1, this part of the SW architecture is implemented by means of a legacy SW reused from previous projects. We were able to reuse it with any modifications thanks to the modularity provided by the ROS environment.

5.5.1 Navigation SW

The robot should be able to map the environment, localize in it and move in desired position while avoiding obstacles. ROS implements a library called Navigation Stack suitable for this purpose. The Navigation Stack is fairly simple on a conceptual level. It takes information from odometry and sensor streams and outputs velocity commands to be sent to a mobile base.

5.5.1.1 Prerequisites

The Navigation SW works if the following pre-requisites are satisfied:

1) It requires a planar laser mounted somewhere on the mobile base. This laser is used for map building and localization.
2) It is meant for both differential drive and holonomic wheeled robots only. It assumes that the mobile base is controlled by sending desired velocity commands to implement in the form of: x velocity, y velocity, theta velocity.
3) It needs odometry information
4) It needs the robot to set up all the transformation between the reference systems of the various sensors and actuators (TF)

- **Laser Data**

The Laser Data are published on the topic /scan, further details about these Data are described in section 4.3.4.

- **Odometry**

The Odometry is calculated by the odometry publisher and data are published on the topic /odom of type ”nav msgs/Odometry” [10]. The Node takes as input < Vx, Vy, Vω > by subscribing to the topic /vel and computes the pose < X, Y, θ > of the robot w.r.t the initial position using reverse kinematics using the formulas below reported:

$$\delta\theta = \omega * \delta t$$

$$\delta x = (Vx * \cos\theta - Vy * \sin\theta) * \delta t$$

$$\delta y = (Vx * \sin\theta + Vy * \cos\theta) * \delta t$$

e:

$$\theta = \theta + \delta\theta$$

$$X = X + \delta x$$

$$Y = Y + \delta y$$

where δ is the difference between one time interval passed between one measure and the precedent one.

- **TF**

At an abstract level, a transform tree [11] defines offsets in terms of both translation and rotation between different coordinate frames. It is used to have a single reference point rather than multiple sensors’ reference points. In our case we need a way of transforming the laser scan we have received from the ”base laser” frame to the ”base link” frame. In essence, we need to define a relationship between the ”base laser” and

”base link” coordinate frames. The TF were already present in the code, but the position of lasers was checked because in long time they could have changed a little their positions.

5.5.1.2 Robot Mapping

The first thing to get is the map of the environment. To do so we use a ROS library called *gmapping* [12]. This library implements a SLAM (Simultaneous Localization and Mapping) using the Rao-Backwellized Particle Filter. This approach takes as input raw laser data and odometry data, builds a variable number of possible maps (each one is called a particle). The best particle is selected taking in consideration the present and past data, as the robot moves around the environment (controlled by human input) the particles will grow similarly and the map will be more precise.

The *gmapping* package is driven by the definition of appropriate parameters. Here, some choices for the value of the parameters are reported; the selection was done by doing multiple tests and selecting the ones that seemed to provide better more precise results.

- *map update interval*: 0.2 - How long (in seconds) between updates to the map.
- *particles:* 100 - Number of particles in the filter

The other default parameters were acceptable for the expected behaviour.

5.5.1.3 AMCL

Once we have a map the robot must localize itself in it; this is done using the ROS library AMCL [13]. AMCL is a probabilistic localization system for a robot moving in 2D. It implements the adaptive (or KLD-sampling) Monte Carlo localization approach, which uses a particle filter to track the pose of a robot on a known map.

Also the *AMCL* package is driven by the definition of appropriate parameters. Here, some choices for the value of the parameters are reported; the selection was done by doing multiple tests and selecting the ones that seemed to provide more precise results. The tests were done both by giving the robot the same velocity commands, and by going around randomly. To understand the meaning of the parameters and find help in tuning them, ROS wiki and this document [66] were used. Unfortunately, not all parameters were sufficiently explained, so some of them were tuned by multiple trials.

Overall Parameters:

- *min particles*: 100
- *max particles*: 1000

This range of number of particles was found satisfying because it is a good balance between the possibility to explore more solutions and a good speed in processing the algorithm.

- *kld err:* 0.05 Maximum error between the true distribution and the estimated distribution.
- *kld z:* 0.95 Upper standard normal quantile for (1 - p), where p is the probability that the error on the estimated distribution will be less than kld err.

There is some probability in the estimation error of the distribution of particles, so the default values were modified to let the algorithm consider some noisy data (given by some random measurements from laser data and inaccuracies in odometry calculation).

- *update_min_d:* 0.05 Translational movement required before performing a filter update. (in meters)
- *update_min_a:* 0.05 Rotational movement required before performing a filter update (in π value radians)

Since the robot can move also by a little, a very frequent filter update is necessary, so 5 cm and 5 radians was found a good value.

Laser Model Parameters:

- *laser_max_beams:* 30 How many evenly-spaced beams in each scan to be used when updating the filter.
- *laser_z_hit:* 0.8 The weight for the probability of a correct measurement
- *laser_z_rand:* 0.2 The weight for the probability of random measurement
- *laser_likelihood_max_dist:* 4.0 The maximum distance to trust laser reads
- *laser_model_type:* "likelihood field"

These parameters describe how AMCL uses laser data. The laser model type represents the type of algorithm chosen between beam, likelihood field, or likelihood field prob (same as likelihood field but incorporates the beamskip feature, if enabled). Likelihood field was chosen because it seems to perform better in this kind of environment.

For the choice of the weight, we decided to be more pessimistic than the default value because AIRLab has irregular walls (full of boxes) that are frequent to cause random measurements because the irregular reflection of laser beams. This is also typical of many scenes in a theatre.

Odometry model parameters:

- *odom_model type:* "omni-corrected"

- *odom_alpha1:* 0.2 Specifies the expected noise in odometry rotation estimate from the rotational component of the robot's motion.
- *odom_alpha2:* 0.5 Specifies the expected noise in odometry rotation estimate from translational component of the robot's motion.
- *odom_alpha3:* 0.5 Specifies the expected noise in odometry translation estimate from
- *odom_alpha4:* 0.5 Specifies the expected noise in odometry translation estimate from the rotational component of the robot's motion.
- *Odom_alpha5:* 0.8 Translation-related noise parameter.

These parameters describe how AMCL uses odometry data. The odometry model is omnidirectional (omni-corrected is used because it solves some bugs). The other parameters are set in a more pessimistic way w.r.t the default ones (they are a little higher), because we found that the omnidirectional odometry model is inherently noisy w.r.t differential because when the robot moves the wheels are not only rolling but also slithering, so a less predictable friction on the ground may cause some issues.

5.5.1.4 Move_Base

The *move_base* [14] package provides an implementation of an action that, given a goal in the world, will attempt to reach it with a mobile base. The *move_base* node links together a global and local planner to accomplish its global navigation task. The *move_base* node also maintains two cost maps, one for the global planner, and one for a local planner that are used to accomplish navigation tasks.

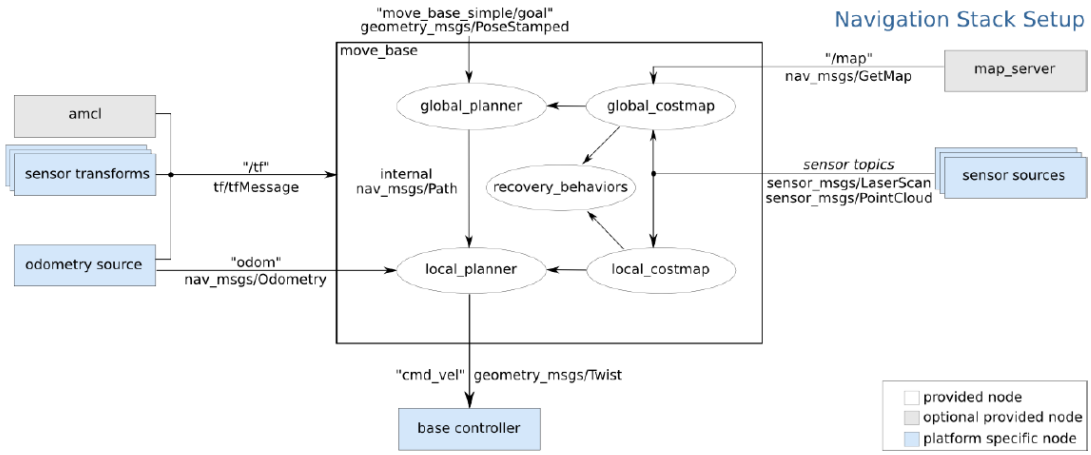

Figure 39: *Move_Base* Architecture

Running the *move_base* node on a robot that is properly configured results in a robot that will attempt to achieve a goal pose with its base within a user-specified tolerance. In the absence of dynamic obstacles, the *move_base* node will eventually get within this tolerance from its goal, or signal failure to the user.

5.5.1.5 Motors Control: Nova Core

The Nova Core board run a ROS node and publish and subscribe topics. This node is in charge of controlling the velocities of the 3 motors. It subscribes to the topic /cmd_vel of type geometry msgs/Twist [21] where is contained the information on the desired velocities < Vx; Vy; ! <. The node computes the kinematics and then directly controls the velocity of each motor. It then checks the real velocity of each motor using the encoders, computes the values of Vx; Vy; ! and publishes this information on the topic /vel.

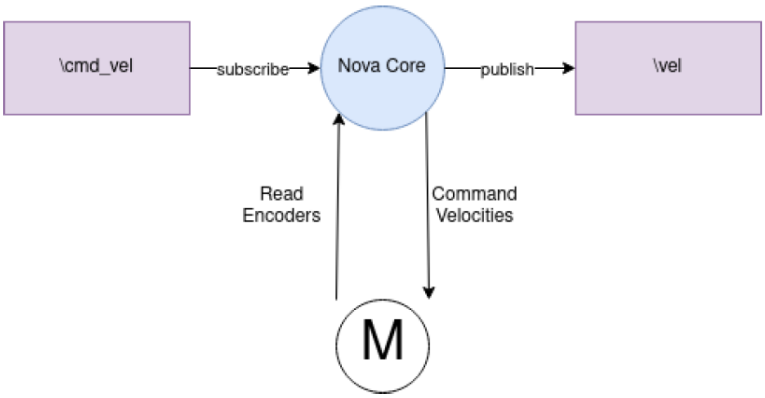

Figure 40: Nova Core node and published/subscribed topics

5.5.1.6 Lasers

Two urg node [22] are launched. These nodes read the laser data from the serial port and publish it on two topics: scan right and scan left. The two data are then merged by another node: laserscan multi merger, which subscribes to the two laser's topics and merges the data into the topic scan. This topic represents the scan value in an area spanning 360 degrees.

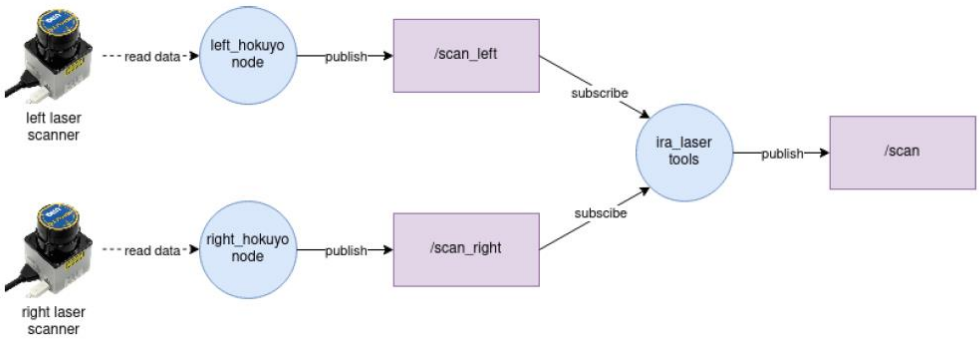

Figure 41: Urg_Nodes and published/subscribed topics

5.5.1.7 Cmd_Vel Manager

If we want to move the robot without the localization we can publish directly on the topic /cmd_vel. To do so, we have implemented the action server: cmd vel manager. The structure of the action is the following: goal: contains the movements to perform response true is all the movements are done, false otherwise current movement the feedback about what movement the robot is performing The goal contains a multiple of 7 float numbers, that indicates all the movements to perform in sequence. The first three number represents the absolute value of the amount of space in meters to move in the x axis, y axis and angle θ (in radians). The second three numbers represent the velocities on Vx, Vy, ω, the sign of the velocities indicates the direction. For example if we want to move from position (0,0,0) to position (-2.0, 5.0,0) by velocity 1.0 m/s we will send the numbers: [2.0,5.0,0.0,- 1.0,1.0,0.0]. The last number represents the amount of time to wait before performing the next movement., which this node is in charge of making it

5.5.1.8 Joystick

5.5.1.8.1 Joy

ROS provides a package called "joy" [26] that contains the joy node, a node that interfaces a generic Linux joystick to ROS. This node publishes a "Joy" message, which contains the current state of each one of the joystick's buttons and axes.

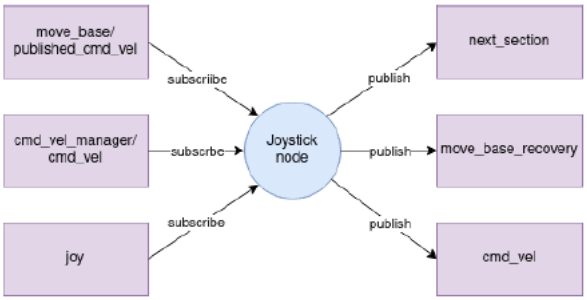

Figure 42: Arduino node and published/subscribed topics

5.5.1.8.2 Joystick Node

The "joystick node" subscribes to the topic "joy", to read the commands coming from the operator. It subscribes to topics /move_base/published_cmd_vel and /cmd_vel_manager/cmd_vel. When "Enable autonomous movement" is pressed it republishes the commands of these 2 topics on /cmd vel, otherwise the joystick node publishes a "zero command" on "cmd vel" and the robot is stopped. For safety reasons, in fact, the robot can only autonomously move when it is enabled by the operator. This is to prevent the robot from hitting a person or falling off the stage. To move the robot "by hand" the two analog sticks are used, the left for linear velocities, the right for angular. To not accidentally move the robot, the "Enable manual movement" button has to be pressed while using the analog sticks. To change the robot's velocities the operator can press Increase Linear Velocity, Decrease Linear Velocity, Increase Angular Velocity or Decrease Angular Velocity. When is required, the operator can order the robot to go to next section by pressing the Next Section Button, then the joystick node publishes "True" on the /next_section topic of type "std msgs/Bool" [24]. When is required, the operator can decide to let the robot retry to use *move_base* by pressing the "Retry Move Base" button or skip to the next section by pressing the "Skip Move Base" Button, then the joystick node publishes "1" or "2" on the /move_base_recovery" topic of type std msgs/Int8MultiArray [23].

5.5.2 Robot's Body and Eyes Control

5.5.2.1 Arduino Node

The Arduino board run a ROS node and publish and subscribe topics. This node is in charge of controlling the battery state, the eyes movement and the body movement. On setup it moves the eyes and the body in the normal position. Every 10 seconds it checks the battery state with the circuit described earlier, if it is too low it plays a buzzer to communicate to the operator to change the batteries as soon as possible. It subscribes also to the topics /arduino/eyes and /arduino/body of type std msgs/Int8MultiArray [23] to receive commands for the eyes and body movements. It is able to move multiple

motors simultaneously. When the servomotors are in the desired position it publishes "True" on topics /arduino/eyes response and /arduino/body response of type std msgs/Bool [24]. We have used the ROS library: rosserial arduino [25], but we have reduced the amount of space occupied by the message queues because it was causing malfunctions. A general schema of the Arduino node is reported in the figure below.

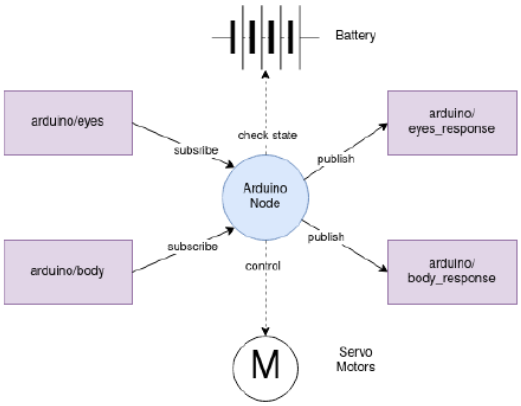

Figure 43: Arduino node and published/subscribed topics

5.5.2.2 Eyes Manager and Body Manager Nodes

These nodes are in charge of moving eyes and body of the robot. They work in a similar way, by implementing an action server. They subscribe to the related topics to receive the commands and publish on the topics to communicate when they have finished processing the request. The structure of the action is the following:

goal: contains the array of positions and speed written on the script

response: true if all the movements are done, false otherwise

current_movement: contains the feedback of what movement the robot is performing (the number).

When called, the action server starts to perform all the required movements. It publishes one movement of the topic /arduino/eyes or /arduino/body and waits for the completion by publishing to the topic /arduino/eyes_response or /arduino/body_response. Between every movement, this node is responsible of making the required pause.

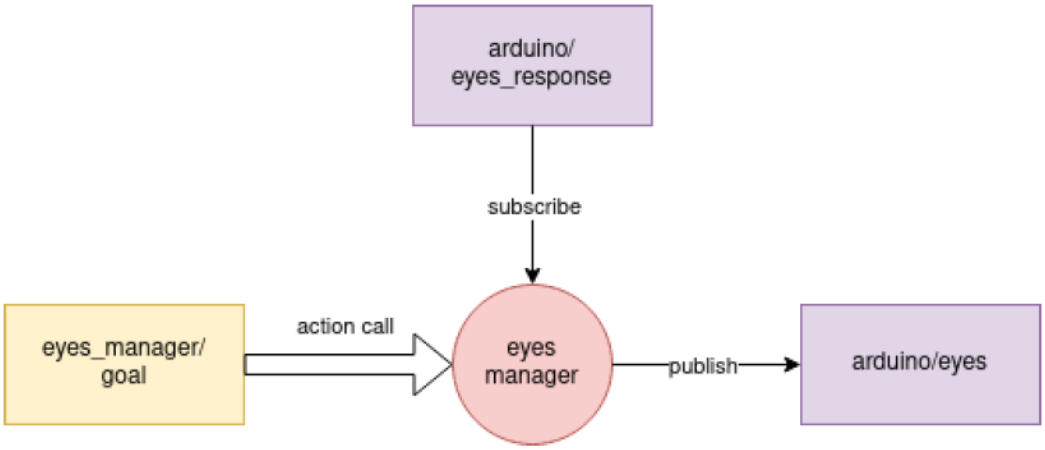

Figure 44: Eyes Manager action server and published/subscribed topics

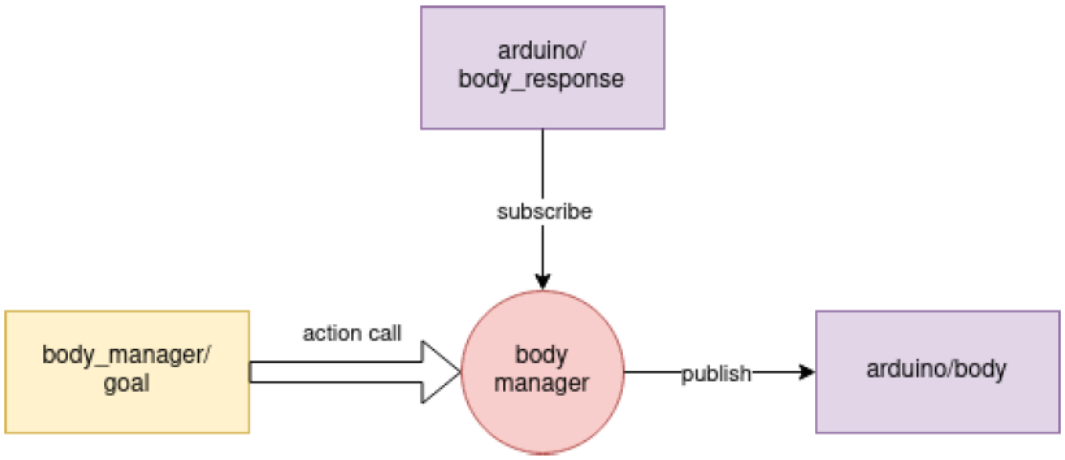

Figure 45: Body Manager action server and published/subscribed topics

6. Test Cases

6.1 Test Strategy

Since the Actor Observation SW, developed by our colleague graduating student, was not yet available at the start of our testing session, we decided to implement a dedicated "simulation SW" able to generate the messages which should come from the observation SW.

This approach has the following additional advantages:

1) It allows to perform a great number of tests in a simple, fast and efficient way.
2) It allows to separate the problems and bugs to be identified and corrected in our SW from the ones that could possibly be present in the Actor Observation SW.
3) It allows to avoid the ambiguities that could possibly exist in the behavior of a real human actor.
4) It allows to perform the test of our SW in a completely repeatable way. This is point is very important because one of our goals is to demonstrate that the robot's behavior can be different even when submitted to exactly the same inputs and initial conditions.

As described in section 5.3.2, the information regarding the actor's status is published on two ROS topics: /scenic_action and /actor_position:

- On the topic /scenic_action is published the scenic action identified by the action recognition module, in the set of pre-defined scenic actions listed in section 3.2.
- On the topic /actor_position the relative position of the actor with respect to the robot is published, both in a "categorical" form ("forward", "behind", "right", "left", "near", "far") and in an "analytical-numerical" form (X and Y coordinates in the robot's reference system).

The interaction with the Actor Observation SW is simulated using a JSON file that contains information coherent with what could actually be performed by an actor, organized organically with respect to the ROS messages expected to be received by the

Behavior Model. This information is read by our simulation stub which feeds it to our operational SW through the two topics described above.

The result of the tests were analyzed by:

- Observing the actual behavior of the robot controlled by our SW
- Producing textual logbooks of the information elaborated by the main nodes of our SW (Actor Interface, Behavior Model Server, Reaction Server and Main Controller) during the execution of the test

In the following sections we report an example of test case and the logs of two different "test runs" executed under the same initial conditions and the same input information.

6.2 Test Case Example

The scenic actions performed by the actor were realized under the hypothesis of a coherent reaction of the robot in the case of the robot's personality being "Commander" and "Surprise" being the initial state. Distances with respect to the robot on the X and Y axis are hypothetical too, but coherent with the scenic action performed by the actor.

6.3 Logbook of Test Run 1

ACTOR'S INFO	BEHAVIOR MODEL	REACTION SERVER	MAIN CONTROLLER
	loading COMMANDER personality state NFA initial emotional state is SURPRISE		

{
"actorPosX": 0.5,
"actorPosY": 0.0,
"scenicAction": "grudge"
}
	transition: surprise, grudge -> anger
	ROBOT'S REACTION: grudge
		REACTING 'grudge'
		approach: -1, rotate: -1
		actions: ['eyes_up']
			world model:
			actorPosX: 0.5
			actorPosY: 0.0
			scenicAction: "grudge"
			robotEmoState: "anger"
			robotReaction: "grudge"
			endScene: False
			distance: 0.5
			angle: 0.0°
			MOVE EYES: [0, 0, 0]
			MOVE BODY: [0, 0, 0]
			MOVE: [0, 0, 3.141592653589793, 0, 0, 1.0, 0]
			current position -> [0, 0]
			MOVE: [0.5625, 0, 0, 0.5, 0, 0, 0]
			current position -> [0.5625, 0]
			PERFORMING REACTION: eyes_up
			MOVE EYES: [0, 4, 1]

{	transition: anger,	REACTING 'disappointment'	world model:

"actorPosX": -1.0, "actorPosY": -1.0, "scenicAction": "running_away" }	running_away -> sadness ROBOT'S REACTION: disappointment	approach: -1, rotate: -1 actions: ['raising_eyes']	actorPosX: -1.0 actorPosY: -1.0 scenicAction: "running_away" robotEmoState: "sadness" robotReaction: "disappointment" endScene: False distance: 1.4142135623730951 angle: -2.356194490192345° MOVE EYES: [0, 0, 0] MOVE BODY: [0, 0, 0] MOVE: [0, 0, 0.7853981633974483, 0, 0, 1.0, 0] current position -> [0.5625, 0] MOVE: [0.0, 0, 0, 0.4, 0, 0, 0] current position -> [0.5625, 0] PERFORMING REACTION: raising_eyes MOVE EYES: [0, 4, 0, 1, 0, 3, 1, 4, 0, 1, 0, 3]
{ "actorPosX": 1.0, "actorPosY": 0.0, "scenicAction": "scolding" },	transition: sadness, scolding -> sadness ROBOT'S REACTION: sharing_sadness	REACTING 'sharing_sadness' approach: 1, rotate: 1 actions: ['bow_slow', 'crossing_eyes_slow']	world model: actorPosX: 1.0 actorPosY: 0.0 scenicAction: "scolding" robotEmoState: "sadness" robotReaction: "sharing_sadness" endScene: False distance: 1.0 angle: 0.0° MOVE EYES: [0, 0, 0] MOVE BODY: [0, 0, 0] MOVE: [0, 0, 0.0, 0, 0, 1.0, 0] current position -> [0.5625, 0] MOVE: [0.75, 0, 0, 0.3, 0, 0, 0] MOVE RECOVERY: [0.5625, 0, 0, - 1, 0, 0, 1] current position -> [0.75, 0] PERFORMING REACTION: bow_slow MOVE BODY: [0, 1, 4] PERFORMING REACTION: crossing_eyes_slow MOVE EYES: [0, 5, 5]

{
"actorPosX": 0.5,
"actorPosY": 0.0,
"scenicAction": "attack"
}
	transition: sadness, attack
	-> sadness
	ROBOT'S REACTION:
	disappointment
		REACTING 'disappointment'
		approach: -1, rotate: -1
		actions: ['raising_eyes']
			world model:
			actorPosX: 0.5
			actorPosY: 0.0
			scenicAction: "attack"
			robotEmoState: "sadness"
			robotReaction: "disappointment"
			endScene: False
			distance: 0.5
			angle: 0.0°

			MOVE EYES: [0, 0, 0] MOVE BODY: [0, 0, 0] MOVE: [0, 0, 3.141592653589793, 0, 0, 1.0, 0] current position -> [0.75, 0] MOVE: [0.5625, 0, 0, 0.4, 0, 0, 0] MOVE RECOVERY: [0.75, 0, 0, -1, 0, 0, 1] current position -> [0.5625, 0] PERFORMING REACTION: raising_eyes MOVE EYES: [0, 4, 0, 1, 0, 3, 1, 4, 0, 1, 0, 3]

{
"actorPosX": 0.5,
"actorPosY": -1.0,
"scenicAction": "surprise"
}
	transition: sadness, surprise -> sadness
	ROBOT'S REACTION: sharing_sadness
		REACTING 'sharing_sadness'
		approach: 1, rotate: 1
		actions: ['bow_slow', 'crossing_eyes_slow']
			world model:
			actorPosX: 0.5
			actorPosY: -1.0
			scenicAction: "surprise"
			robotEmoState: "sadness"
			robotReaction: "sharing_sadness"
			endScene: False
			distance: 1.118033988749895
			angle: -1.1071487177940904°
			MOVE EYES: [0, 0, 0]
			MOVE BODY: [0, 0, 0]
			MOVE: [0, 0, 1.1071487177940904, 0, 0, -1.0, 0]
			current position -> [0.5625, 0]
			MOVE: [0.8385254915624212, 0, 0, 0.3, 0, 0, 0]
			MOVE RECOVERY: [0.5625, 0, 0, -1, 0, 0, 1]
			current position -> [0.8385254915624212, 0]
			PERFORMING REACTION: bow_slow
			MOVE BODY: [0, 1, 4]
			PERFORMING REACTION: crossing_eyes_slow
			MOVE EYES: [0, 5, 5]

{
"actorPosX": 0.0,
"actorPosY": 1.0,
"scenicAction": "sharing_happiness”
}
	transition: sadness, sharing_happiness -> joy
	ROBOT'S REACTION: sharing_happiness
		REACTING 'sharing_happiness'
		approach: 1, rotate: 1
		actions: ['self_rotate', 'dangling_eyes']
			world model:
			actorPosX: 0.5
			actorPosY: -1.0
			scenicAction: "surprise"
			robotEmoState: "sadness"
			robotReaction: "sharing_sadness"
			endScene: False
			distance: 1.118033988749895
			angle: -1.1071487177940904°

			MOVE EYES: [0, 0, 0] MOVE BODY: [0, 0, 0] MOVE: [0, 0, 1.1071487177940904, 0, 0, -1.0, 0] current position -> [0.5625, 0] MOVE: [0.8385254915624212, 0, 0, 0.3, 0, 0, 0] MOVE RECOVERY: [0.5625, 0, 0, -1, 0, 0, 1] current position -> [0.8385254915624212, 0] PERFORMING REACTION: bow_slow MOVE BODY: [0, 1, 4] PERFORMING REACTION: crossing_eyes_slow MOVE EYES: [0, 5, 5]

6.4 Logbook of Test Run 2

ACTOR'S INFO	BEHAVIOR MODEL	REACTION SERVER	MAIN CONTROLLER
	loading COMMANDER personality state NFA initial emotional state is SURPRISE		

{
"actorPosX": 0.5,
"actorPosY": 0.0,
"scenicAction": "grudge"
}
	transition: surprise, grudge -> anger
	ROBOT'S REACTION: grudge
		REACTING 'grudge'
		approach: -1, rotate: -1
		actions: ['eyes_up']
			world model:
			actorPosX: 0.5
			actorPosY: 0.0
			scenicAction: "grudge"
			robotEmoState: "anger"
			robotReaction: "grudge"
			endScene: False
			distance: 0.5
			angle: 0.0°
			MOVE EYES: [0, 0, 0]
			MOVE BODY: [0, 0, 0]
			MOVE: [0, 0, 3.141592653589793, 0, 0, 1.0, 0]
			current position -> [0, 0]
			MOVE: [0.5625, 0, 0, 0.5, 0, 0, 0]
			current position -> [0.5625, 0]
			PERFORMING REACTION: eyes_up
			MOVE EYES: [0, 4, 1]

{ "actorPosX": -1.0, "actorPosY": -1.0, "scenicAction":	transition: anger, running_away -> anger ROBOT'S REACTION: attack	REACTING 'attack' approach: 1, rotate: 1 actions: ['bow']	world model: actorPosX: -1.0 actorPosY: -1.0 scenicAction: "running_away"

"running_away" }			robotEmoState: "anger" robotReaction: "attack" endScene: False distance: 1.4142135623730951 angle: -2.356194490192345° MOVE EYES: [0, 0, 0] MOVE BODY: [0, 0, 0] MOVE: [0, 0, 2.356194490192345, 0, 0, -1.0, 0] current position -> [0.5625, 0] MOVE: [1.125, 0, 0, 0.9, 0, 0, 0] MOVE RECOVERY: [0.5625, 0, 0, -1, 0, 0, 1] current position -> [1.125, 0] PERFORMING REACTION: bow MOVE BODY: [0, 1, 1]
{ "actorPosX": 1.0, "actorPosY": 0.0, "scenicAction": "scolding" },	transition: anger, scolding -> anger ROBOT'S REACTION: intimidate	REACTING 'intimidate' approach: 1, rotate: 1 actions: ['bow']	world model: actorPosX: 1.0 actorPosY: 0.0 scenicAction: "scolding" robotEmoState: "anger" robotReaction: "intimidate" endScene: False distance: 1.0 angle: 0.0° MOVE EYES: [0, 0, 0] MOVE BODY: [0, 0, 0] MOVE: [0, 0, 0.0, 0, 0, 1.0, 0] current position -> [1.125, 0] MOVE: [0.9, 0, 0, 0.6, 0, 0, 0] MOVE RECOVERY: [1.125, 0, 0, -1, 0, 0, 1] current position -> [0.9, 0] PERFORMING REACTION: bow MOVE BODY: [0, 1, 1]

{
"actorPosX": 0.5,
"actorPosY": 0.0,
"scenicAction": "attack"
}
	transition: anger, attack -> anger
	robot's reaction: attack
		REACTING 'attack'
		approach: 1, rotate: 1
		actions: ['bow']
			world model:
			actorPosX: 0.5
			actorPosY: 0.0
			scenicAction: "attack"
			robotEmoState: "anger"
			robotReaction: "attack"
			endScene: False
			distance: 0.5
			angle: 0.0°
			MOVE EYES: [0, 0, 0]
			MOVE BODY: [0, 0, 0]
			MOVE: [0, 0, 0.0, 0, 0, 1.0, 0]
			current position -> [0.9, 0]
			MOVE: [0.45, 0, 0, 0.9, 0, 0, 0]
			MOVE RECOVERY: [0.9, 0, 0, -1, 0, 0, 1]

			current position -> [0.45, 0] PERFORMING REACTION: bow MOVE BODY: [0, 1, 1]

{
"actorPosX": 0.5,
"actorPosY": -1.0,
"scenicAction":
"surprise"
}
	transition: anger, surprise -> anger
	robot's reaction: attack
		REACTING 'attack'
		approach: 1, rotate: 1
		actions: ['bow']
			world model:
			actorPosX: 0.5
			actorPosY: -1.0
			scenicAction: "surprise"
			robotEmoState: "anger"
			robotReaction: "attack"
			endScene: False
			distance: 1.118033988749895
			angle: -1.1071487177940904°
			MOVE EYES: [0, 0, 0]
			MOVE BODY: [0, 0, 0]
			MOVE: [0, 0, 1.1071487177940904, 0, 0, -1.0, 0]
			current position -> [0.45, 0]
			MOVE: [1.0062305898749055, 0, 0, 0.9, 0, 0, 0]
			MOVE RECOVERY: [0.45, 0, 0, -1, 0, 0, 1]
			current position -> [1.0062305898749055, 0]
			PERFORMING REACTION: bow
			MOVE BODY: [0, 1, 1]
			WAITING FOR BEHAVIOR MODEL RESPONSE

{
"actorPosX": 0.0,
"actorPosY": 1.0,
"scenicAction":
"sharing_happiness”
}
	transition: anger, sharing_happiness -> joy
	robot's reaction: sharing_happiness
		REACTING 'sharing_happiness'
		approach: 1, rotate: 1
		actions: ['self_rotate', 'dangling_eyes']
			world model:
			actorPosX: 0.0
			actorPosY: 1.0
			scenicAction: "sharing_happiness"
			robotEmoState: "joy"
			robotReaction: "sharing_happiness"
			endScene: True
			distance: 1.0
			angle: 1.5707963267948966°
			MOVE EYES: [0, 0, 0]
			MOVE BODY: [0, 0, 0]
			MOVE: [0, 0, 1.5707963267948966, 0, 0, 1.0, 0]
			current position -> [1.0062305898749055, 0]
			MOVE: [0.75, 0, 0, 0.75, 0, 0, 0]
			MOVE RECOVERY: [1.0062305898749055, 0, 0, -1, 0, 0, 1]
			current position -> [0.75, 0]
			PERFORMING REACTION: self_rotate
			MOVE: [0, 0, 6.28, 0, 0, 1, 0]
			current position -> [0.75, 0]

			PERFORMING REACTION: dangling_eyes MOVE EYES: [0, 1, 0, 1, 2, 0, 1, 1, 0, 1, 2, 0, 0, 0, 0]

6.5 Test Evaluation

During the execution of the test it was possible to verify that:

- The evolution of the robot's emotional state and of the robot's reactions were coherent with the possible transitions defined by the state-transition function associated to the selected personality.
- The transitions actually performed were different between the two test runs, in coherence with the non-deterministic behaviour implemented by the Behaviour Model NFA.
- The various steps of the computed reaction were in line with the expected ones.
- The movements of the robot's base, body and eyes were in line with the reactions steps generated by the Reaction Server and actuated by the Main Controller via the robot's control SW.

7. Conclusions and Future Developments

7.1 Conclusions

The aim of this study was to implement in a robot the capability to "emotionally" interact with a human actor. Our work allowed us to:

- combine phases when the robot follows a predefined "script" with phases where the robot reacts in real-time with the actions performed by a human actor on a stage;
- simulate an "internal emotional life" of the robot by means of a simple, yet rigorous formal method like the finite-state automaton;
- transfer concepts like "emotions" and "personality" from the realm of Psychology to the field of Robotics, exploiting the instruments provided by the theory of abstract machines;
- verify that the FSA formalism, even if conceptually simple, allows to set-up an "improvisational" context, and has the possibility to be expanded to higher levels of complexity which will increase the capabilities of the robot to perform a "spontaneous" behaviour.

We performed the first step of complexity enhancement in the frame of our study, passing from a deterministic FSA to the implementation of a non-deterministic FSA. This simple step greatly enhanced the "unpredictability" of the robot's behaviour, making it look less "mechanic" and more "natural". Many other possibilities exist to further exploit the possibilities offered by this conceptual framework, that we were not able to fully develop but are briefly described in section 7.2.5.

7.2 Future Works

7.2.1 Integration with Actor Observation SW

The first topic to be completed is the integration of the SW we have developed with the one which is object of a parallel graduation thesis and is in charge of observing the human actor and recognizing its attitude, as stated in sections 1.1 and 5.3.

We agreed with our colleague working on the observation SW a very simple interface mechanism (a couple of ROS topics publishing messages in a standard String format), therefore there shouldn't be any difficulty under the technical point of view.

The content of the message includes the position of the actor and the scenic action he is performing, which can be one in a list of possibilities defined together with our colleague (see section 3.2 above).

In any case, our SW implementing the robot's emotional life and reactions is independent from this list: in case a new or different scenic action is defined, it would be sufficient to update the JSON files defining the various personalities to implement the new behaviour without modifying a single line of code.

7.2.2 Adopting the ROS-2 Framework

As explained in section 5.2.4, it is some years that the ROS community is moving from ROS-1 to ROS-2 framework. However, in the scope of our work it was not possible to transfer all the legacy SW developed under ROS-1 in the new environment. Therefore we decided develop our SW in ROS-1 and to include the ROS Bridge in our architecture so that it could interface with SW developed according to the new standard.

Migrating the overall SW architecture in ROS-2 should be the object of a future work, to be carried on without urgency (as ROS-1 "Noetic" version will be completely supported and maintained at least up to the end of 2025) but with the aim to harmonize the complete product with the modern trend.

Given the complexity of the legacy SW in charge of controlling to robot's HW platform, which is made up by several concurrent nodes with a considerable number of interfaces (topics, services, actions etc.), the migration should be performed adopting a step-by-step approach: the various functional modules should be moved under ROS-2 one by one, connecting the ported SW to the rest of legacy SW by means of the ROS Bridge package. The availability under ROS-2 of stable versions for all the ROS-1 library packages (such as AMCL and move_base) and the possibility to access them via ROS Bridge should also be assessed.

7.2.3 Enhancing the Expressive Possibilities of the HW Platform

The Robocchio product, previously developed at AIRlab as explained in section 4, has been a good platform to experiment our approach to emotions expressions and improvisation. However, its physical possibilities to express emotions are somewhat limited: basically, it can only navigate around the stage, tilt its body and move its flexible antennae with eyes.

While making our experiments, we had the idea that the possibility to express emotions could have been significantly enhanced by conceptually simple modifications, e.g. the addition of “eyebrows” and “eyelids” to the eyes. These additions could be easily realized by means of a 3-D printer and could be moved dedicated mechanisms connected to small motors (see figures 44 and 45).

We lacked the time to fully implement this concept, but we made anyway some sketches using the Blender 3-D graphic designing tool:

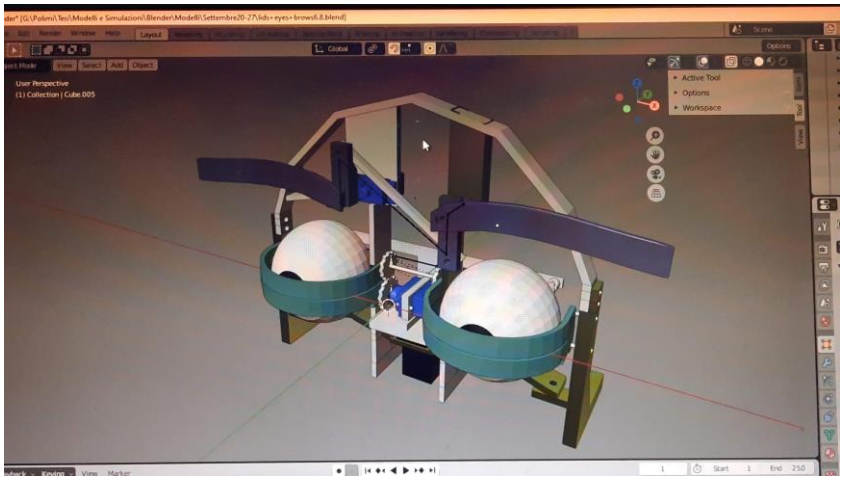

Figure 46: Eyelids 3-D design with Blender (1)

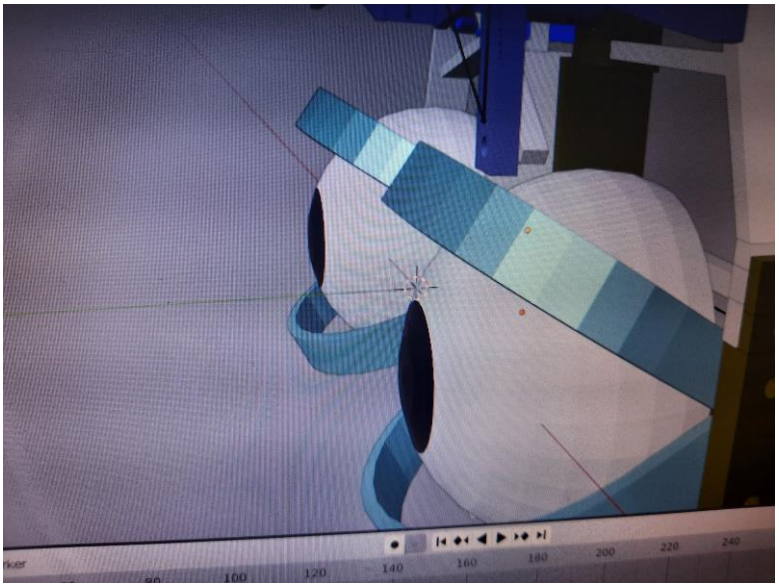

Figure 47: Eyelids 3-D design with Blender (2)

7.2.4 Using Audio and Speech

As stated in section 3.1, both the recognition and the expressions of emotions are mainly based on gestures. Nevertheless, the addition of sounds and speech can of course greatly expand the possibilities of recognizing and performing emotional behaviour.

7.2.4.1 Playing Audio Files

The PC embedded in the TriskarOne platform is naturally equipped with speakers and the capability to play audio files (e.g. by means of the standard Python library package PyAudio [27]).

Therefore, emitting beeps, sounds or pre-recorded words and phrases could be added to the "actions" specified in the predefined Scripts and to the possibilities listed in tables 2 and 3 for expressing emotions and performing reactions.

7.2.4.2 Recognizing Speech

Many off-the-shelf speech recognition packages exist, especially in Python which is one of the two main languages used for SW development in the ROS environment. Any one of these packages can be used to acquire words spoken by the actor and to transform them into text files which can then be submitted to further processing.

There are practically no limits to the possibilities offered by natural language recognition software, from the simple identification of specified keywords to AI-based analysis of the actor's attitude and intention. For example, the more basic option (identification of a list of words which can be associated to emotions) could be used in parallel from the input of the scenic action recognition package to drive the evolution of the robot emotional state, adding more nuances to our Behaviour Model. Alternatively, the actor's speech could be added as a "trigger" for an action of a predefined Script (see section 5.4.3.4) and then be linked to the execution of a Script action playing and audio file, implementing a kind of "dialogue" between actor and robot (see next section).

7.2.4.3 Implementing Answering Capabilities

The recognition of actor's words could be linked to the playing of audio files containing sounds or text synthesized by means of standard text-to-speech library packages like Sonantic [17] or Readspeaker [18]. In this way it could be possible to implement a feedback loop in our simulated behaviour which would greatly enhance the "naturalness" of the robot performance.

7.2.5 Increasing the Complexity of the Behaviour Model

During our work we had many ideas concerning the possibilities to enhance the robot's behaviour simulation capabilities, which could be exploited using the baseline architecture that we have described in this document.

Unfortunately, we were not able to pursue them in the available time frame, but in our opinion they could be the object of interesting developments which could be carried-on in dedicated future studies. In the next sections we present these ideas and some hints on how they could be realized.

7.2.5.1 Implement different reactions according to Personality

In section 3.4 we described how the reaction of the robot is defined as a function of the scenic action and of the reached emotional state. In our implementation this function is valid for all the possible "personalities" of the robot; however, in principle also this function can be made dependent from the robot's personality, implementing a different reaction table for any given personality (as we have done for the transition function between emotional states, see section 3.3.4).

7.2.5.2 Modelling Emotional States Evolution using Fuzzy Logic

Instead of assigning a definite value to the robot Emotional State, we could associate to each state a "membership function" as defined in the Fuzzy Logic Theory. This means that:

1) the statement "Robocchio is angry" would not have a simple True or False value, but would be associated to a function expressing the "degree of truth" of the statement;
2) the robot would not necessarily be in a single emotional state, but could be associated to more emotional states with a different "degree of truth" for any association. For example, it could be "very surprised" and "a bit sad".

As a consequence, the transition function δ() described in section 3.3.3 would not depend only on the identified scenic action and the current emotional state, but should have as inputs all the robot's possible states and the value of the membership function associated to each state. The probability to reach a new emotional state should therefore be calculated on the basis of the values of all the membership functions.

Moreover, the transition function should determine not a single emotional state to be reached but the set of new values for the membership function of all the possible states.

This process would of course be much more similar to the human behaviour, where emotions are rarely defined in a sharp way. Its implementation would not present significant technical difficulties; however, the trial phase could become very complex

because the number of possible reactions to a given scenic action would rise exponentially.

7.2.5.3 Modelling Robot's Personality using Fuzzy Logic

In section 3.3.2 we explained the reasons why we decided to model the robot's personality using the MBTI method. However, an alternative could be to use the "Big Five" method: in this case, instead of having a fixed number (16) of personalities, each one associated to a transition function in the NFA driving the evolution of the emotional state, we would have a continuous spectrum of personalities defined by the points selected in the five axis corresponding to the 5 pairs of opposed personality functions (see Figure 1).

As a consequence, the transition function δ() described in section 3.3.3 would not depend only on the identified scenic action and the current emotional state (or on the "membership functions" associated to each possible state, see previous section) but would also depend on the value of the 5 "membership functions" associated to each one of the "Big Five" personality traits (Openness, Consciousness, Extroversion, Agreeableness and Neuroticism).

Moreover, we could also make the membership functions to the five traits *dependent on time*, to make the robot change its attitude as it spends more time interacting with the human actor. For example, to model the robot "getting acquainted" with the actor, we could make the values of Extraversion, Agreeableness and Openness increase linearly with time (maybe at different slopes), while keeping Consciousness stable and making Neuroticism change randomly.

Of course, in this way the transition function would require a multi-dimensional mathematical analysis package (e.g. numpy) to be calculated, and the trials would be even more complex than described in the previous section.

Appendix A

A.1 – Source Code

The source code is available at https://github.com/Nicole4597/robo_Sim.

A.2 – Example of Script

```
{
"section1": {
    "trigger":"after_precedent",
    "trigger_data": -1,
    "actions": {
        "move_base":[-0.19,2.21,0.33,0.94],
        "move_eyes": [
            3,3,
            0,3,
            4,3,
            0,5,
            3,2,
            0,6
        ]
    }
},
"section2": {
    "trigger":"after_precedent",
    "trigger_data": -1,
    "actions": {
        "start_scene": -1
    }
},
"section3": {
    "trigger": "after_precedent",
    "trigger_data": 3,
    "actions": {
        "move_body":[0,3,2,3,0,3,1,3,0,3,3,4,0,4,4,3,0,4],
        "move_base":[-0.37,-2.07,0.25,0.96]
    }
},
"section4": {
    "trigger":" after_command",
    "trigger_data": -1,
    "actions": {
        "end": -1
    }
}
}
```

A.3 – JSON file for "Architect (INTJ)" Personality

```
{
	"anger" : [
		["sharing_sadness", "surprise" , 0.25],
		["sharing_sadness", "anger" , 0.75],
		["sharing_fear", "surprise" , 0.25],
		["sharing_fear", "anger" , 0.75],
		["sharing_happiness", "surprise" , 0.5],
		["sharing_happiness", "anger" , 0.5]
	],

	"fear" : [
		["sharing_sadness", "sadness" , 0.25],
		["sharing_sadness", "fear" , 0.75],
		["surprise", "surprise", 0.1],
		["surprise", "fear", 0.9],
		["sharing_happiness", "surprise" , 1]
	],

	"joy" : [
		["attack", "anger" , 0.2],
		["attack", "surprise" , 0.8],
		["intimidate", "anger" , 0.2],
		["intimidate", "surprise" , 0.8],
		["sharing_fear", "joy" , 0.5],
		["sharing_fear", "surprise" , 0.5],
		["sharing_sadness", "joy" , 0.5],
		["sharing_sadness", "surprise" , 0.5]
	],

	"sadness" : [
		["attack", "fear" , 1],
		["intimidate", "fear" , 1],
		["sharing_happiness", "joy" , 0.1],
		["sharing_happiness", "surprise" , 0.7],
		["sharing_happiness", "sadness" , 0.2]
	],

	"surprise" : [
		["attack", "anger" , 0.2],
		["attack", "fear" , 0.8],
		["intimidate", "anger" , 0.2],
		["intimidate", "fear" , 0.8],
		["sharing_sadness", "sadness" , 0.5],
		["sharing_sadness", "surprise" , 0.5]
	]
}
```

A.4 – JSON file for "Advocate (INFJ)" Personality

```
{
	"anger" : [
		["sharing_happiness", "joy", 0.1],
		["sharing_happiness", "surprise", 0.9],
		["sharing_sadness", "sadness", 0.5],
		["sharing_fear", "sadness", 0.5],
		["sharing_sadness", "anger", 0.5],
		["sharing_fear", "anger", 0.5],
		["surprise", "surprise", 0.25],
		["surprise", "anger", 0.75],
		["attack", "fear", 0.2],
		["attack", "anger", 0.8],
		["intimidate", "fear", 0.2],
		["intimidate", "anger", 0.8]
	],

	"fear" : [
		["surprise", "surprise", 1],
		["sharing_sadness", "sadness", 1],
		["disappointment", "sadness", 0.5],
		["disappointment", "fear", 0.5],
		["sharing_happiness", "joy", 1],
		["happy_person", "joy", 0.5],
		["happy_person", "fear", 0.5]
	],

	"joy" : [
		["attack", "anger", 0.2],
		["attack", "fear", 0.8],
		["intimidate", "anger", 0.2],
		["intimidate", "fear", 0.8],
		["sharing_fear", "fear", 1],
		["surprise", "surprise", 1],
		["sharing_sadness", "sadness", 1],
		["disappointment", "sadness", 0.5],
		["disappointment", "joy", 0.5],
		["running_away", "sadness", 0.5],
		["running_away", "joy", 0.5]
	],

	"sadness" : [
		["sharing_happiness", "joy", 0.5],
		["sharing_happiness", "sadness", 0.5],
		["surprise", "joy", 0.5],
		["surprise", "sadness", 0.5],
		["attack", "fear", 0.8],
		["intimidate", "fear", 0.8],
		["attack", "anger", 0.2],
		["intimidate", "anger", 0.2],
		["sharing_fear", "fear", 0.5],
		["sharing_fear", "sadness", 0.5]
	],
```

```
	"surprise" : [
		["attack", "fear", 0.8],
		["attack", "anger", 0.2],
		["intimidate", "fear", 0.8],
		["intimidate", "anger", 0.2],
		["sharing_fear", "fear", 0.8],
		["sharing_fear", "surprise", 0.2],
		["happy_person", "joy", 0.8],
		["happy_person", "surprise", 0.2],
		["sharing_happiness", "joy", 1],
		["sharing_sadness", "sadness", 1]
	]
}
```

A.5 – JSON file for “Commander (ENTJ)” Personality

```
{
      "anger" : [
      ["sharing_happiness", "surprise", 0.5],
      ["sharing_happiness", "joy", 0.5],
      ["sharing_sadness", "surprise", 0.25],
      ["sharing_fear", "surprise", 0.25],
      ["sharing_sadness", "anger", 0.75],
      ["sharing_fear", "anger", 0.75],
      ["running_away", "sadness", 0.5],
      ["disappointment", "sadness", 0.5],
      ["running_away", "anger", 0.5],
      ["disappointment", "anger", 0.5]
],

"fear" : [
      ["attack", "anger", 0.5],
      ["attack", "fear", 0.5],
      ["intimidate", "anger", 0.5],
      ["intimidate", "fear", 0.5],
      ["scolding", "anger", 0.5],
      ["scolding", "fear", 0.5],
      ["sharing_happiness", "surprise", 0.75],
      ["sharing_happiness", "joy", 0.25],
      ["surprise", "surprise", 0.25],
      ["surprise", "fear", 0.75],
      ["sharing_sadness", "sadness", 0.5],
      ["sharing_sadness", "fear", 0.5]
],

"joy" : [
      ["attack", "anger", 0.1],
      ["attack", "surprise", 0.15],
      ["attack", "fear", 0.75],
      ["grudge", "anger", 0.1],
      ["grudge", "surprise", 0.15],
      ["grudge", "fear", 0.75],
      ["intimidate", "anger", 0.1],
      ["intimidate", "surprise", 0.15],
      ["intimidate", "fear", 0.75],
      ["sharing_fear", "fear", 0.75],
      ["sharing_fear", "surprise", 0.25],
      ["running_away", "surprise", 0.5],
      ["running_away", "joy", 0.5]
],

"sadness" : [
      ["sharing_happiness", "joy", 0.75],
      ["sharing_happiness", "surprise", 0.25],
      ["surprise", "joy", 0.5],
      ["surprise", "sadness", 0.5]
],
```

```
"surprise" : [
      ["sharing_sadness", "sadness", 0.5],
      ["sharing_sadness", "surprise", 0.5],
      ["sharing_happiness", "joy", 1],
      ["happy_person", "joy", 0.5],
      ["happy_person", "surprise", 0.5],
      ["surprise", "joy", 0.5],
      ["surprise", "surprise", 0.5],
      ["attack", "anger", 0.5],
      ["attack", "fear", 0.5],
      ["grudge", "anger", 0.5],
      ["grudge", "fear", 0.5],
      ["intimidate", "anger", 0.5],
      ["intimidate", "fear", 0.5]
]
}
```

Bibliography

[1] url: https://www.softbankrobotics.com/emea/en/nao

[2] url: https://us.aibo.com]

[3] url: https://beatbots.net/keepon-pro. [accessed 15-Marc-2021]

[4] url: https://design.ros2.org/articles/why_ros2.html

[5] url: http://wiki.ros.org/noetic

[6] url: https://index.ros.org/p/ros1_bridge

[7] url:http://wiki.ros.org/actionlib

[8] url:http:/www1.shuttleeuproductsdiscontinuedbarebonesdh310/

[9] url:https://store.arduino.cc/arduino-uno-rev3

[10] url:http:/docs.ros.orgenmelodicapinav_msgshtmlmsgOdometry.html

[11] url:https://wiki.ros.org/navigation/Tutorials/RobotSetup/TF

[12] url:http://wiki.ros.org/gmapping

[13] url: http://wiki.ros.org/amcl

[14] url: http://wiki.ros.org/move_base

[15] url: http://design.ros2.org/articles/changes.html

[16] url: https://design.ros2.org/articles/ros_middleware_interface.html

[17] url: https://www.sonantic.io

[18] url:https://www.readspeaker.com/solutions/text-to-speech-software

[19] url: https://design.ros2.org/articles/ros_on_dds.html

[20] url: https://design.ros2.org/articles/Node_to_Participant_mapping.html

[21] url:http://docs.ros.org/en/melodic/api/geometry_msgs/html/msg/Twist.html

[22] url:http://wiki.ros.org/urg_node

[23] url:http:/docs.ros.orgenmelodicapistd_msgshtmlmsg/Int8MultiArray.html

[24] url:http://docs.ros.org/en/melodic/api/std_msgs/html/msg/Bool.html

[25] url: http://wiki.ros.org/rosserial_arduino/Tutorials/Arduino

[26] url: http://wiki.ros.org/joy

[27] url: https://pypi.org/project/PyAudio/

[28] Lorenzo Bonetti: “Design and Implementation of an Actor Robot for a Theatrical Play”.
Graduation Thesis (A.A. 2019-2020)

[29] Mark Chen, John A. Bargh: “Consequences of Automatic Evaluation: Immediate Behavioral Predispositions to Approach or Avoid the Stimulus” (1999)
https://doi.org/10.1177/0146167299025002007

[30] Julian Angel-Fernandez: “TheatreBot: Studying emotion projection and emotion enrichment system for autonomous theatrical robot".
PhD thesis (Sept. 2016).
doi: 10.13140/RG.2.2.11709.46567

[31] Janine Willis and Alexander Todorov: [“First Impressions: Making Up Your Mind After a 100-Ms Exposure to a Face”.
doi:10.1111/j.1467-9280.2006.01750]

[32] Andrea Bonarini, Stefano Boriero, and Ewerton Lopes Silva de Oliveira: “Robot player adaptation to human opponents in physical, competitive robogames".
In: 2020 29th IEEE International Conference on Robot and Human Interactive Communication (RO-MAN). IEEE. 2020, pp. 851-856

[33] Cynthia Breazeal et al.: “Interactive robot theatre".
doi: 10.1109/IROS.2003.1249722

[34] Albert van Breemen: “Animation engine for believable interactive user-interfacerobots"
doi: 10. 1109/IROS.2004.1389845

[35] A. Bruce, J. Knight, S. Listopad, B. Magerko and I. R. Nourbakhsh, "Robot improv: using drama to create believable agents”.
In: Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065), 2000, pp. 4002-4008 vol.4.
doi: 10.1109/ROBOT.2000.845355. [36]

[37] Paul Ekman: “An Argument For Basic Emotions".
In: Cognition & Emotion 6 (May 1992), pp. 169-200.
doi: 10.1080/02699939208411068

[38] url: https://en.wikipedia.org/wiki/Emotion.

[39] Wargo, E.: "How many seconds to a first impression?"
The Observer.19 (2006)

[40] Naomi Fitter: “What's the Deal with Robot Comedy?”, 2020.
url: https://spectrum.ieee.org/whats-the-deal-with-robot-comedy

[41] Michael Gielniak and Andrea Thomaz: “Enhancing interaction through exaggerated motion synthesis".
In: Mar. 2012, pp. 375-382. isbn: 978-1-4503-1063-5.
doi: 10.1145/2157689.2157813

[42] Guy Hoffman: “OpenWoZ: A Runtime-Configurable Wizard-of-Oz Framework for Human-Robot Interaction" (Mar. 2016)

[43] Guy Hoffman, Rony Kubat, and Cynthia Breazeal: “A hybrid control system for puppeteering a live robotic stage actor".
In: Sept. 2008, pp. 354-359. isbn:f978-1-4244-2212-8.
doi: 10.1109/ROMAN.2008.4600691

[44] Guy Hoffman and Gil Weinberg: “Interactive Improvisation with a Robotic Marimba Player".
In: Auton. Robots 31 (Oct. 2011), pp. 133-153.
doi: 10.1007/978-3-642-22291-7_14

[45] Mehrabian, A., & Ferris, S. R.: “Inference of attitudes from nonverbal communication in two channels” (1967).
In: Journal of Consulting Psychology, 31(3), pp. 248–252.
https://doi.org/10.1037/h0024648

[46] Franz Kafka: “La Metamorfosi” (1915)

[47] Heather Knight and Reid Simmons: “Expressive motion with x, y and theta: Laban Effort Features for mobile robots".
In: vol. 2014. Aug. 2014, pp. 267-273.
doi: 10.1109/ROMAN.2014.6926264

[48] Rudolf Laban.: “Modern Educational Dance” (1963)

[49] Naumann, L. P.; Vazire, S.; Rentfrow, P. J.; Gosling, S. D.: "Personality Judgments Based on Physical Appearance".
In: Personality and Social Psychology Bulletin. 35 (12): 1661–1671. (17 September 2009)
doi:10.1177/0146167209346309

[50] Sheng Fang, Catherine Achard, Séverine Dubuisson: “Personality classification and behaviour interpretation: an approach based on feature categories” (October 2016)
doi:10.1145/2993148.2993201

[51] Oshin Vartanian, Keith Stewart, David R. Mandel, Nada Pavlovic, Lianne McLellan, Paul J. Taylor: “Personality assessment and behavioural prediction at first impression”.
https://doi.org/10.1016/j.paid.2011.05.024.

[52] Izabella Pluta: “Teatro e robotica: os androides de Hiroshi Ishiguro, em encenações de Oriza Hirata".
In: 3 (May 2016), pp. 65-79.
doi: 10.36025/arj.v3i1.8405

[53] Cynthia Vinney: “Understanding the Big Five Personality Traits” (2018).
url: https://www.thoughtco.com/big-five-personality-traits-4176097

[54] Oliver P. John, Sanjay Srivastava: “The Big Five Trait taxonomy: History, measurement, and theoretical perspectives” (1999).
In L. A. Pervin & O. P. John (Eds.), Handbook of personality: Theory and research (pp. 102–138), Guilford Press.
url: https://psycnet.apa.org/record/1999-04371-004

[55] “Roboscopie, the robot takes the stage!” (2011)
url: https://www.openrobots.org/wiki/roboscopie

[56] Krisztina Rosner “The gaze of the robot: Oriza Hirata's robot theatre”
url: https://thetheatretimes.com/the- gaze- of- the- robot

[57] John M. Grohol: “The Big Five Personality Traits” (2019).
url: https://psychcentral.com/lib/the-big-five-personality-traits

[58] Giuseppe Luigi Leandro Russo and Andrea Bonarini: “Model for social human-robot interactions and application to the theatrical context" (2016)

[59] Courtney E. Ackerman: “Big Five Personality Traits: The OCEAN Model Explained” (2021).
url: https://positivepsychology.com/big-five-personality-theory/

[60] Michael Suguitan and Guy Hoffman: “Blossom: A Handcrafted Open-Source Robot".
In: ACM Transactions on Human-Robot Interaction 8 (Mar. 2019), pp. 1-27.
doi: 10.1145/3310356

[61] Louis Tassinary, John Cacioppo, and Gary Berntson: “Handbook of Psychophysiology” (2017)

[62] John P. Oliver, Christopher J. Soto: “The next Big Five Inventory (BFI-2): Developing and assessing a hierarchical model with 15 facets to enhance bandwidth, fidelity, and predictive power.”
url: https://psycnet.apa.org/record/2016-17156-001

[63] Kerry L. Jang, W. John Livesley, Philip A. Vemon: “Heritability of the Big Five Personality Dimensions and Their Facets: A Twin Study” (1996).
url: https://onlinelibrary.wiley.com/doi/10.1111/j.1467-6494.1996.tb00522.x

[64] Karl R. Wurst: “I Comici Roboti: Performing the Lazzo of the Statue from the Commedia Dell'Arte".
In: AAAI Mobile Robot Competition, 2002

[65] Garth Zeglin et al: “HERB's Sure Thing: A rapid drama system for rehearsing and performing live robot theater".
In: Proceedings of IEEE Workshop on Advanced Robotics and its Social Impacts, ARSO 2015 (Jan. 2015), pp. 129-136.
doi: 10.1109/ARSO.2014.7020993

[66] Kaiyu Zheng: “ROS Navigation Tuning Guide", June 2017

[67] Paul Costa et al.: “Gender Differences in Personality Traits Across Cultures: Robust and Surprising Findings” (2001).
url:https://www.researchgate.net/publication/11825676_Gender_Differences_in_Personality_Traits_Across_Cultures_Robust_and_Surprising_Findings

[68] Laura L. Yoder: “The Relationship Between Personality Type and Marital Satisfaction” (2011).
url:https://www.researchgate.net/publication/304125074_The_Relationship_Between_Personality_Type_and_Marital_Satisfaction

[69] Michael Gurven et al.: “How Universal Is the Big Five? Testing the Five-Factor Model of Personality Variation Among Forager-Farmers in the Bolivian Amazon” (2012).
url:https://www.researchgate.net/publication/233939049_How_Universal_Is_the_Big_Five_Testing_the_Five-Factor_Model_of_Personality_Variation_Among_Forager-Farmers_in_the_Bolivian_Amazon

[70] url: https://www.iit.it/web/icub-tech

[71] I. B. Myers: “The Myers-Briggs Type Indicator: Manual”.
Consulting Psychologists Press, 1962.

Acknowledgements

I would like to thank Professor Bonarini who gave me the opportunity to carry out this research work and to complete my academic career in an interesting and innovative way.

I also thank all those who have worked in AIRlab in recent years and have prepared the ground on which I have been able to walk in these months. We are always part of a team, even when it seems like we are working on our own, because our every result is based on the efforts of others who have worked before us. In particular I thank Lorenzo Bonetti, whose work is an integral part of the project I have realized.

Finally, I thank my fellow student Lorenzo Farinelli, who carried out the other part of the project with which my software interfaces, and who has been of great help to me in the difficulties that have arisen in my work.