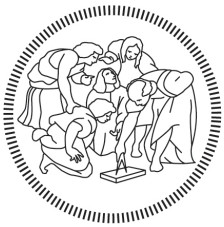
POLITECNICO DI MILANO
Corso di Laurea Magistrale in Ingegneria Informatica
Dipartimento di Elettronica, Informazione e Bioingegneria



POLITECNICO
MILANO 1863

# Design and Implementation of an Actor Robot for a Theatrical Play

AI & R Lab
Laboratorio di Intelligenza Artificiale
e Robotica del Politecnico di Milano

Relatore: Prof. Andrea Bonarini

Tesi di Laurea di:
Lorenzo Bonetti, matricola 920400

Anno Accademico 2019-2020

*Ai miei genitori*

# Abstract

Human-robot interaction is the field of robotics that studies the various ways in which humans and robots can communicate. The robot can improve its communication through the expression of emotions, bringing it closer to the way humans communicate. Theatre is the perfect testing ground for understanding how robots can communicate by displaying emotions. In fact, the theatre is a place where you can decide a priori what actions to carry out, but without detaching yourself from reality.

This document explains how an actor robot was designed and implemented both in its physical aspect and it capability to perform in a theatrical play. It is capable of expressing emotions through the movement of the eyes and pelvis and moving along the stage. The robot is programmed to follow a script containing all the lines and the actions that it must perform to play its role and express its emotions; it is largely autonomous in interpreting its part.

Finally, a video was shot in which the robot plays a character and was evaluated by an audience through a questionnaire. The results show that the emotions that the robot is supposed to show are recognizable and that the audience appreciates the robot in a real theatrical performance.

# Sommario

L'interazione uomo-robot è quel campo della robotica che studia i vari modi in cui uomini e robot possono comunicare. Il robot può migliorare la sua comunicazione attraverso l'espressione delle emozioni, avvicinandolo al modo di comunicare degli esseri umani. Il teatro è il campo di prova perfetto per comprendere come i robot possono comunicare mostrando delle emozioni. Infatti, il teatro è un luogo dove si può decidere a priori che azioni svolgere, ma senza distaccarsi dalla realtà.

Questo documento spiega come è stato progettato e costruito un robot attore sia nell'aspetto che nelle capacità di recitare in uno spettacolo. Esso è capace di esprimere delle emozioni attraverso il movimento di occhi e bacino e spostandosi lungo il palcoscenico. Il robot è programmato per seguire un copione contenente tutte le azioni che il robot deve svolgere per interpretare il suo ruolo ed esprimere le proprie emozioni; è in gran parte autonomo nell'interpretare la sua parte.

Infine, è stato girato un video in cui il robot interpreta un personaggio ed è stato valutato da un pubblico attraverso un questionario. I risultati mostrano che le emozioni che il robot dovrebbe mostrare sono riconoscibili e che il pubblico apprezzerebbe il robot in un vero spettacolo teatrale.

# Ringraziamenti

Ringrazio inanzitutto il professor Andrea Bonarini, per essere stato sempre disponibile a consigliarmi idee e soluzioni, ma anche per il supporto morale nei momenti di difficoltà. Ringrazio anche l'ing. Giulio Fontana e tutti i tesisti e dottorandi presenti in Airlab per l'utilissimo supporto tecnico. Ringrazio i miei genitori per aver sempre creduto in me anche nei momenti di sconforto. Ringrazio i miei colleghi e amici Alex, Federica, Raffaella e Vittorio per aver permesso che il mio percorso universitario fosse gioioso. Ringrazio i miei amici Andrea, Corinne Francesca e Luigi che da sempre sono una spalla su cui contare.

# Contents

# List of Figures

# List of Tables

# Chapter 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 more and more in contact, not only in the industries but also in hospitality, health care, public spaces, and smart homes. In the future, it will not be an unusual thing to see a robot serving at the restaurant or in a reception of a hotel. These new usages of robots require them to communicate with humans in a more "humane" way, not only with text lines and computer-generated voice, but also by emphasizing the interaction by showing emotional expressions, as 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 emotional expression. Despite this, theatre is not completely detached from the real world, it must be truthful to be effective. The robot in the theatre does not have to be intelligent, but it must appear so in the same way the actor must make the public think that he is his character. Eugenio Barba, the founder of the Odin Theatre, said: *"I love theatre because I repel fiction"*.

The purpose of our work is to build a robot capable of interpreting a play by interacting with human actors, by showing its emotions to the audience, to entertain them. The robot is autonomous in positioning itself on the stage and in respecting the correct times to act. In the future, this robot can be used for different purposes: in addition to its use on the stage, it can be used to educate young people and adults in robotics or it can be used as a platform to study how to react to human emotions in real-time.

## 1.1 Brief description of the work

Our project was inspired by some works on robots capable of expressing emotions such as the robot-cushion developed by Julian Fernandez and prof. Andrea Bonarini in 2016 [31]. 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 speeds and placing the robot in a context.

Starting from these results we decided to build a robot that could play a theatrical scene. The common factor of these robots is that they are often suited to playing a single character, rather than being able to play multiple roles [66] [65]. Furthermore, robots are often complex and expensive. We, therefore, decided to build a non-humanoid robot capable of having great expressiveness, but being relatively simple to build and program. There are several projects of robot actors in the world, many of which will be illustrated in chapter 2. Some are based on pre-existing industrial robots which however lack expressive capacity. Others are based on humanoid robots [53], very technically advanced, but often not appreciated, because they are too similar to humans (Uncanny Valley [52]). We, therefore, focused on building a robot by taking inspiration from those who play without pretending to be anything else than a robot [44] [45] [65] [66] [44] [56]. Furthermore, the robot is largely autonomous, it must be able to position itself correctly on the stage and enter the correct line of the speech. Human intervention is reduced to a minimum, the operator directs the robot only in delicate moments and reacts in case of unexpected events. The robot will then be able to adapt to different roles. The robot is designed starting from a pre-existing structure, adding movements and functions that allow it to express itself. The robot is also able to speak by reproducing preregistered audio files.

A language to define the script containing the actions to be performed has been defined, and the robot has been programmed to follow the script. The robot can locate itself on the stage and is able to listen to the other actors on the stage to understand when to say his lines. The robot was then programmed to play a character on stage, as a demonstration of its capabilities: Pinocchio (hence the name Robocchio). To get people's feedback, a video was recorded and viewers were asked to fill out a questionnaire. Unfortunately, it was not possible to have a live audience due to the Covid19-restrictions. Our hope is that our robot can be widely used on the stage thanks to its low costs and the wide possibilities of customization that makes it adaptable to different contests. Furthermore, the robot will be used as a platform to test emotions by reacting to the human interlocutor.

## 1.2 Structure of the Thesis

The document is structured as follows:

- **Chapter2** describes some theoretical notions on emotions and theater and the state of the art regarding emotional and theatrical robots.
- **Chapter3** generally describes the project and the system requirements.
- **Chapter4** describes the structure of the robot from the mechanical point of view, the moving parts and the external appearance.
- **Chapter5** describes the software and framework used and illustrates the basic concepts necessary for understanding the following chapters.

- **Chapter6** describes the used sensors and actuators .
- **Chapter7** describes how the planner was set up to move the robot base around the stage.
- **Chapter8** describes how the robot manages to speak and follow speech.
- **Chapter9** details the software architecture that controls the robot.
- **Chapter10** illustrates how the operator can control the robot remotely if needed.
- **Chapter11** describes how the play script is structured.
- **Chapter12** describes how the robot displays emotions.
- **Chapter13** describes how the various parts of the robot were tested.
- **Chapter14** describes the play that the robot interprets and the various stylistic choices.
- **Chapter15** collects the conclusions on the accomplished work and the possible future works.

**Appendix**

- **Appendix A** - Description of every package of the workspace and their content.
- **Appendix B** - User Manual: how to set up the robot and how to recover from possible. failures.
- **Appendix C** - The script used for the play.

# Chapter 2

# Theoretical Framework

## 2.1 Emotions

### 2.1.1 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 [39]". Living beings use emotions for self-preservation [62], 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 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 the emotions and feelings of other species, for example, dogs and humans are proved to communicate even without using words.

For thousands of years many scientists and philosophers developed theories about the mind's mechanisms that drive emotions and tried to classify them. In the next paragraph some of the most relevant ones are reported.

#### Discrete Model

The Discrete model is a theory developed by Paul Ekman and his colleagues' in 1990 [38], 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, 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.

### Circumplex Model

The circumplex model of affect, developed by Russel [58], is a dimensional theory that organizes emotions in a Cartesian plan, such that each axis represents an attribute associated to an emotion. One of the most used models is the one suggested by Russell that organizes emotional states in two dimensions: valence and arousal. Valence is related to how pleasant emotion is, ranging from negative to positive, whereas arousal indicates the energy of emotion. The distribution of emotions in Russel's model is reported in figure 2.1

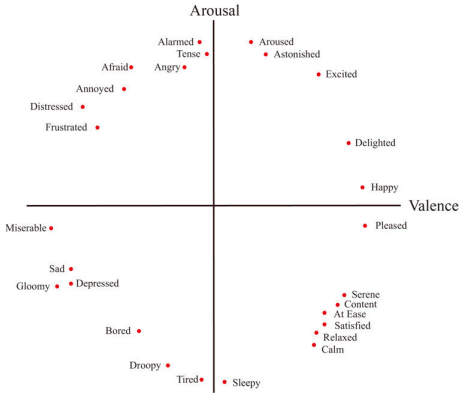



Figure 2.1: Russel Circumplex Model [58]

### Plutchik Theory

In 1980, Robert Plutchik [54] diagrammed a wheel of eight emotions: joy, trust, fear, surprise, sadness, disgust, anger and anticipation. Plutchik also theorized twenty-four "Primary", "Secondary", and "Tertiary" dyads (feelings composed of two emotions). The wheel of emotions can be paired in four groups:

- Primary dyad = one petal apart = Love = Joy + Trust.
- Secondary dyad = two petals apart = Envy = Sadness + Anger.
- Tertiary dyad = three petals apart = Shame = Fear + Disgust.
- Opposite emotions = four petals apart = Anticipation $\notin$ Surprise.

There are also triads, emotions composed of 3 primary emotions. This leads to a combination of 24 dyads and 32 triads, making 56 emotions at 1 intensity level. Emotions can be mild or intense: for example, distraction is a mild form of surprise, and rage is an intense form of anger. The schema of Plutchik Theory is reported in Figure 2.2

### Tomkinks Theory

Tomkins' theory [63] integrates various perspectives. For Tomkins, the affect system evolved to solve the problem of the overwhelming information to which individuals are exposed. His theory says that people cannot handle all the information that comes from the outside in the same way, but it is their affect system that chooses what is most important at all times. For example, someone might be completely engrossed

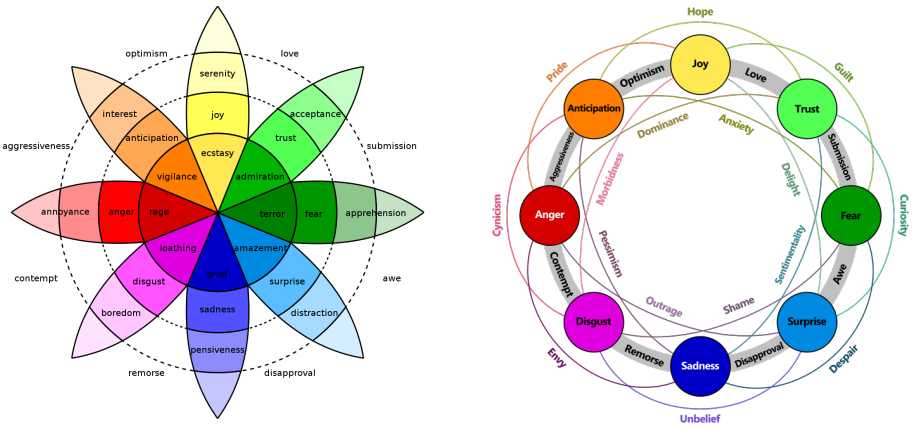



Figure 2.2: Plutchink' Theory: primary, secondary and tertiary triads [54].

in reading a book, but a loud and sudden sound could still warn him. This kind of behaviour could be obtained through the activation of different systems. Tomkins recognized four systems closely related to affect:

- *Pain* is a motivator for very specific events that influences our bodies.
- *Drive* deals with the basic needs that human body could need (e.g., eating, breathing).
- *Cognitive* interprets the world and makes inference from it.
- *Affect* focuses on driving person's attention to specific stimuli.

More importantly, Tomkins suggested that affect in certain situations may cause pain and drive systems to be omitted, while the affect and cognitive systems could work together. Because affection has a main role in human subsistence, he describes nine affects that are triggered by brain activities, and organizes them into three categories:

1. *Positive affects* imply a reward (i.e., Interest-excitement, enjoyment-joy).
2. *Negative affects* are related to situations that may threaten people's well-being (i.e.fear-terror, distress-anguish, anger-rage, disgust and shame).
3. *Neutral affects* do not affect people's behaviour (i.e. surprise-startle).

Finally, Tomkins suggested that these affects are the input of the system that selects an emotion. In his model, each person has an internal "script" that tends to map the affects to specific emotions. This mapping is generated by the personal experience of each person.

### 2.1.2 Emotion Expression

Once described what is an emotion and given some examples of how many of them there are, we want to analyze how a living being expresses emotions and how it is possible to recognize them. Various theories and methods have been proposed; here are reported two of them, mainly focused on the expressions of the emotion on a theatrical stage.

### Laban Movement Analysis

Laban Movement Analysis [49] 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* describes which body parts are moving, and how each part influences the others.
- *Shape* describes the way the body changes during a movement.
- *Space* defines the connection between the movement and the environment.
- *Effort* focuses on changes in the movements w.r.t an inner intention.

**Effort**
Effort is a system for understanding the more subtle characteristics of movement with respect to inner intention. Punching someone and catching an object is very similar in terms of arm extension, but the speed, strength and attention with which this movement is done changes totally. Laban defined effort as related to four different factors, each of which ranges between two boundaries as listed below:

- *Weight* is related to how the body weight is used in the movements, and it ranges from light to strong (delicate vs. powerful).
- *Space* defines the movements as indirect or direct.
- *Time* specifies the velocity during the movement, from quick to sustained.
- *Flow* defines the quality in the movements, which ranges from bound to free.

However, any further details to interpret the symbolic representations of these factors is given to the actor. Being effort related to the inner intentions, it is the most suitable element to describe emotions, which, in the case of acting people, are driven by inner intentions.

### Stanislavski's System

Stanislavski's system [60] is a systematic approach to train actors and help them to fit in their part in the play. It is one of the most famous and utilized methods in theatre schools.
*"When I give a genuine answer to the if, then I do something, I am living my own personal life. At moments like that there is no character. Only me. All that remains of the character and the play are the situation, the life circumstances, all the rest is mine, my own concerns, as a role in all its creative moments depends on a living person, i.e., the actor, and not the dead abstraction of a person, i.e., the role"*

An actor's performance is animated by the pursuit of a sequence of "tasks". A task is a problem, embedded in the "given circumstances" of a scene, that the character needs to solve. This is often framed as a question: "What do I need to make the other person do?" or "What do I want?"

In preparing and rehearsing for a role, actors break up their parts into a series of discrete "bits", each of which is distinguished by the dramatic event of a "reversal point", when a major revelation, decision, or realisation alters the direction of the

action in a significant way. Each "bit" or "beat" corresponds to the length of a single motivation (task or objective). Once the circumstances have been defined actors use the "magic if" to interpret the role. The "Magic if" is the ability of the actor to relate every bit to past experiences in his life, similar to the situation he or she has to interpret (obviously not the same). For example, if the character fails in doing a task and it feels very sad, the actor has to interpret the role thinking of a failure in his life. Thanks to this technique it is easier for the actor to associate the correct emotions to the situation.

## 2.2 Studies on Robot Expressions

We will now focus on studies related to emotional robots. In this section, some concepts about robot animation are introduced and then some examples are presented. Among the many examples of emotional robots, only the ones that we think best represent their categories are reported to not be repetitive.

### 2.2.1 The aspect of the Robot: The Uncanny Valley

The Uncanny Valley [52] is a theory presented by Masashiro Mori that illustrates how the level of appreciation of robots evolves as their resemblance to humans grows (Figure 2.3). According to this theory, starting from industrial robots that are totally non-anthropomorphic, as the robot takes on human characteristics, the affinity with the observer grows. However, when the similarity becomes excessive, there is a sharp decline in appreciation (the uncanny valley), this is because the observer perceives that the robot is artificial despite the great similarity with reality and there is a feeling of disturbance. The curve is amplified when motion is added to the robot. In fact, although the ability to move increases the satisfaction of the observer, movements that are too similar to reality, but which can be perceived as artificial, make the robot look like a zombie, creating a strong repulsion.

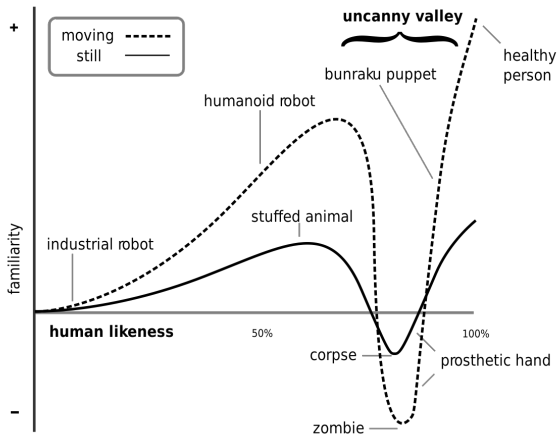



Figure 2.3: The uncanny valley [52]

### 2.2.2 Animate a robot

Tiago Ribeiro and Ana Paiva [55], taking inspiration from the animation principles of Walt Disney designers [46], defined 12 rules to develop expressive robots.

1. *Squash and Stretch*: "The movement and liquidness of an object reflect that the object is alive. One rule of thumb is that despite them changing their form, the objects should keep the same volume while squashing and stretching".

2. *Anticipation*: "Anticipating movements and actions helps viewers and users to understand what a character is going to do".

3. *Staging*: "Staging is related to the general set-up in which the character expresses itself. This principle is related to making sure that the expressive intention is clear to the viewer".

4. *Straight Ahead and Pose-to-Pose*: "Straight ahead action is more of a free method, when the animators know what they want to do, but they have not completely foreseen it, so they start on the first frame and go on sequentially animating on to the last Pose-to-Pose animation is used when the sequence is pre-planned, and it is especially useful when making use of physics and synchronizing movements".

5. *Follow-Through and Overlapping Action*: "This principle works as an opposite of Anticipation. When a character stops doing something, it should not stop abruptly, for that causes an unnatural feeling".

6. *Slow In and Slow Out*: "Along with anticipation and follow-through, objects should not be abrupt when they start or stop moving".

7. *Arcs*: "This principle states that natural motions occur in arcs, so this should be taken into account when designing animation".

8. *Secondary Action*: "This kind of action does not directly contribute to the expression of the character, but aids in making it believable, for example the movement of the body while the robot is speaking.

9. *Timing*: "Animation is, of course, all about timing, it is important to perform the correct actions in the right moment".

10. *Exaggeration*: "Animated characters and objects do not need to follow the rules of our world and our physics. Exaggeration can be used to emphasize the robot's movements, expressions or actions, in order to make them more noticeable and convincing"

11. *Solid Drawing*: "The main concept to get from this principle is asymmetry. Faces rarely exhibit the same expression on the left and right sides, and almost no poses are symmetrical"

12. *Appeal*: "Motion and behaviours of robots should also be easy to understand, because if the users don't understand what they see, their appeal for the robot will fall, for example if we want a viewer or user to love a character, then it should be beautiful and gentle. If we want them to hate a character it should be stiff and grotesque"

### 2.2.3 Examples of Emotional Robots

#### Blossom

Blossom [61] is one of the most recent open-source social robotics platform (Figure 2.6). 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 [43] 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. The exterior of the robot is in textile and can be easily replaced and personalized (one of Hoffmann's statements is: "Why robots should be all the same? Why not defining and making the robot's body, e.g., by knitting it?"). It lacks movement on the ground, because it does not have wheels and cannot interact with people because it does not have sensors.

Figure 2.4: Inside Blossom

Figure 2.5: Possible external aspects of Blossom

Figure 2.6: Blossom [61]

#### iCat

iCat [35] 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 motion coordination and the detection of human expressions, but it has no other possible uses outside the laboratory.

Figure 2.7: iCat [35]

### NAO

Nao [1] is a humanoid robot developed for many different uses (Figure 2.8). Many studies and projects in the emotion field used NAO for demonstrations. For example [41] 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, mirroring the uncanny valley.

Figure 2.8: Nao [41]

### Simon

Simon [42] is a humanoid robot built for studying how different shades of a movement can modify the perception of emotion, in particular, how exaggerated motion in experiments enhances the interaction through the benefits of increased engagement and perceived entertainment value (Figure 2.9). 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.

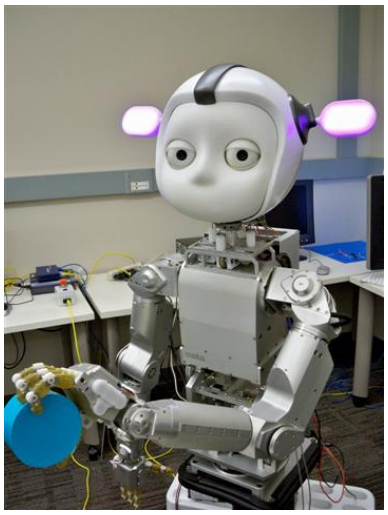


Figure 2.9: Simon [42]

### Keepon

Keepon [3] is a social robot designed for interaction and research with children, developed by Hideki Kozima (Figure 2.10). 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 behavior 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 teleoperator, has interacted with children in schools and remedial centers, as well as with the general public and a simplified version also reached the market as a dancing toy.

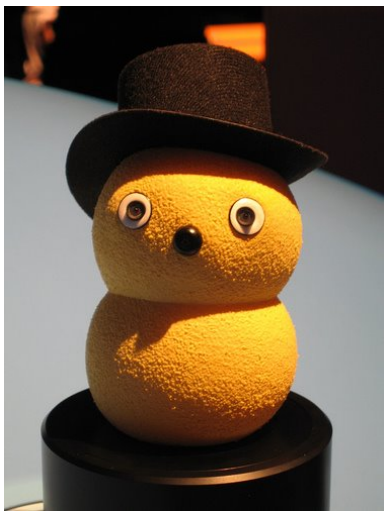


Figure 2.10: Keepon [3]

## 2.3 Theatre

Theatre is considered a lively art; even if it is not a real world, it is suitable to represent situations and experiences that may happen in the real world, but in a more controllable contest. Unlike TV and cinema, theatre is unique in every performance, the differences between one and another can be caused by the actors, who, as humans, do not behave in the same way every day, but also by the public because its reaction may modify the timing and the progress of the performance.

### The Stage

The stage is where all the action happens. It is usually a rectangular space closed by 3 sides and the fourth one facing the public. It could happen that it is not perfectly planar, but can have some irregularities and trapdoors and usually it has a little slope (to let the actors be more visible). On the side in front of the public there is the background, on the other two sides there are the wings from which actors enter and exit. The stage can be conceptually divided into 9 parts as shown in figure 2.11, actors usually take one of these positions. They can look at 8 directions as shown in figure 2.12

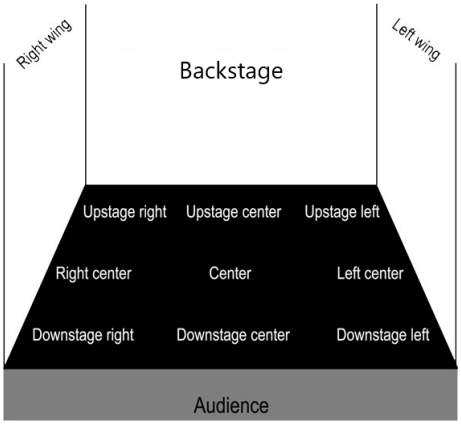



Figure 2.11: Stage division used by directors to give instructions.

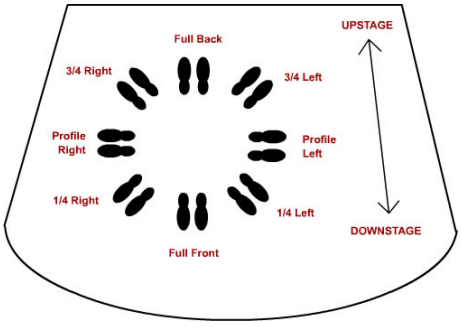



Figure 2.12: Eight possible actors' body positions.

Figure 2.13: Positions and Orientation on the stage

## 2.4 Theatre Robots

Theatre is the perfect test field for emotional robots and many are the different kinds of machines developed to perform theatrical plays. David V. Lu and William D. Smart [51] say that both theatre and HRI aim to replicate some elements of humanity. *"This common aim allows us to exploit the similarity in structure between the two by modeling robot interactions in theatrical terms. Furthermore, HRI and theatre are both inherently interactive. Some might argue that the latter is not actually interactive, since the theatre is scripted, everything is planned out and rehearsed beforehand. Furthermore, there is rarely interaction that crosses the so-called "fourth wall" that separates the actors from the audience. Thus, theatre cannot possibly be a fitting model for social interaction because it lacks interaction. This argument underlines one of the key points about good acting and good interaction: it must be constantly interactive. In both realms, participants must be constantly making adjustments to their actions based on the other's actions, even if all of the high-level actions are scripted/specified in advance. Both will have to adapt to subtle differences in the situations. If the robot does not react to specific circumstances, by not making eye contact or by giving generic responses to questions, the sense that the robot is genuinely interacting with the humans is lost. For an actor, the experience of live theatre must be interactive. Actors who merely "go through the motions", and do not change their behavior in response to others fail to truly interact. Such performances are generally badly reviewed, labeling the actor "robotic". This is precisely the label from which HRI aims to break free by creating interaction that is not "robotic", but engaging."*

The two scientists also underlined what kind of problems the realization of a theatrical robot may have to face:

- **Articulation Problem**: "robots have limited expressivity due to the usually low number of degrees of freedom".
- **Intentionality Problem**: "the robot's intentions are not always clear. Simple motions are often ambiguous, and complex movements do not always give enough consideration to the information the movement transmits to observers".
- **Interpretation Problem**: "recognize the human gestures and contextualize them in the current situation".
- **Validation Problem**: "The human element of the interactions is often unpredictable, making the results of interactions too situation-specific to compare to each other, so there is not an objective way to evaluate the robot's performance".

They finally propose two ways to include human emotions in theatrical robots:

- **Explicit**: Program and select all the movements in advance, before the play.
- **Implicit**: Observe the actor on stage and dynamically select the movements to make.

### Classification of Theatrical robots

David V. Lu [50] proposed a way to classify theatrical robots. The two Cartesian axes in Figure 2.14 represent automation and control:

1. **Automation**: "Differentiates between the complexities of the algorithms needed to run the robot. It allows us to decide who is actually acting". It can be:

   - *Human Produced*: "actions are produced either through teleoperation of the robot, specifying exactly how the robot should move in code (i.e., hard-coding the motions), or using a human's performance to directly generate the robot's performance".
   - *algorithmic*: "systems which generate their behavior via computation without explicit human input".
   - *hybrid*: "systems which have behavior partially specified by the human which can then be modified by an algorithm to achieve some additional behavior, or vice versa, with the human over-riding some algorithmically specified behavior".

2. **Control**: "constant awareness of what others are doing in the scene around them and being able to adjust the own performance to better mesh with them". It can be:

   - *open-loop*:" the robot does the same exact thing in every performance regardless of external factors".
   - *closed-loop*: "the performance is generally the same every time, but has a feedback mechanism which allows the robot to react to the given circumstances".
   - *free*: "during the performance the system is constructed so that the robot can do nearly anything".

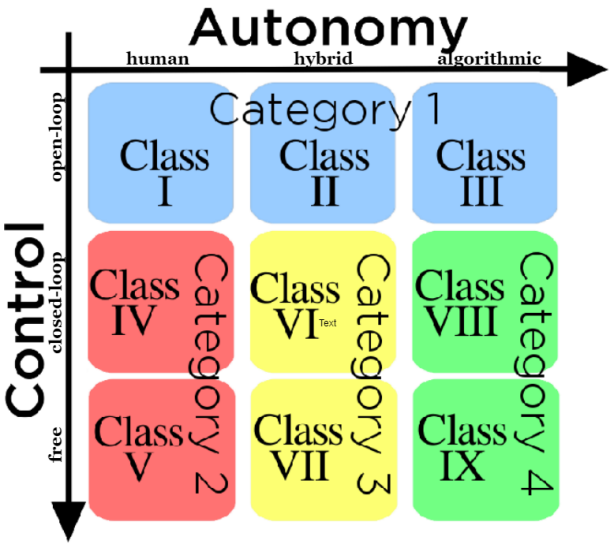



Figure 2.14: Classes and categories in robot ontology [50]

The combination of these characteristics produces 9 classes, from the simplest one (human-produced, open-loop) to the most complex (algorithmic-free controlled), which is not represented by real robots but only by science-fiction. So we can identify 4 categories (Figure 2.14):

1. *Playback*: "this category contains the three classes that are completely open-loop. Some might argue that these are not quite actors (or robots for that matter) because they do not actually react to their environment at all".

2. *Teleoperated*: "systems that integrate human input in a much more direct way. By having a human doing the control during the actual performance (either in person or remotely), the performance becomes more interactive, providing a feedback mechanism and ergo becomes more closed-loop".

3. *Collaborative*: "the third category contains all interactive hybrid systems. The performance combines elements of human control with some degree of machine intelligence".

4. *Autonomous Acting*: "this category features the most autonomous of all the robot actors. These robots' behaviors are generated primarily by algorithms, with little human intervention with exception of perhaps feeding in the source material. The human will still write the algorithm, but the performance will be generated by that algorithm, not explicitly programmed in or inspired by the performance of the human".

**Faults and problems on stage**

The stage, differently from a laboratory, is an environment where unexpected events may happen; for this reason every theatrical robot must be able to recover from any fault that may happen, because "the show must go on". Abrar Fallatah, Jeremy Urann, and Heather Knight [40] studied what are the possible failures that might happen in a theatrical environment:
**Choreographic** (80%):

1. *Timing Failure*: The robot does its actions correctly, but in the wrong moment

2. *Spatial Failure*: The robot is in the incorrect place on the stage

3. *Sequence Failure*: The robot does not perform the correct sequence of actions

**Technical**(20%):

1. *Hardware*

2. *Software*

3. *Sensing Failure*: a sensor gives incorrect readings

4. *Communication Failure*: there is a fault in the communication between the robot and the operator

They proposed general countermeasures for these kinds of failures:

1. Stop the play and restart from the beginning (if it is necessary, rewrite the opera)

2. Stop for some minutes and restart

3. Let the human actor to adapt to the situation

4. Direct human intervention to solve the problem during the performance

### 2.4.1 Examples of theatrical robots

There are many different styles of theatrical plays, different genres, different ways to act and communicate emotions. Similar 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 [53]. 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 [57] "The Metamorphosis" by Franz Kafka [47] was presented, but the main character Gregor Samsa was transformed into a robot instead of a cockroach (Figure 2.15) "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)."

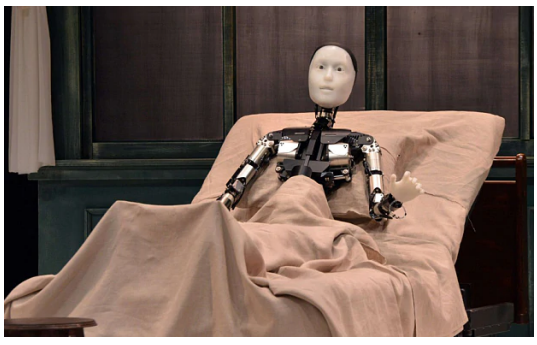


Figure 2.15: Robot performing "The Metamorphosis" [53]

The university of Taiwan developed a humanoid robot able to reproduce facial expressions and body expressions, the face has 23 servo-motors and the arms have 7 D.O.F. 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

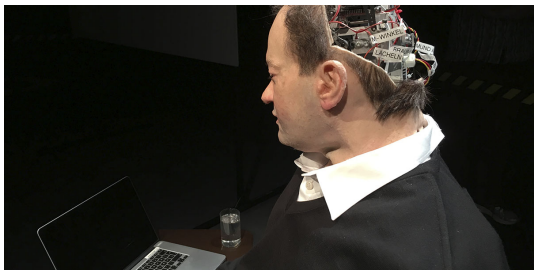


Figure 2.16: The humanoid robot that performed the play: "Uncanny Valley" [4]

behaving like a human. The result shows that even if the public is amazed by the technological progress, is not really interested in the improvements in the performance the robot does w.r.t the human actor.

The theatrical trio: "Rimini Protokoll" presented a show named "Uncanny Valley" [4] where a humanoid robot shaped as the writer Thomas Melle. The performance is a lecture on the bipolarity of the humanoid robot, unstable like a human, but predictable like a machine. The show has the purpose to make the audience ask themselves if humanoid robots will become part of a daily routine. The robot is capable of making fairly faithful limb movements and has a good lip-synch, but remains stationary throughout the play (Figure 2.16).

The general problem with humanoid robots is that they often end up in the uncanny valley. In fact, the level of technology reached by these robots allows them to imitate the human aspect very well, but they can still be perceived as artificial. This is why they risk not being appreciated by the public from an aesthetic point of view, despite the fact that they amaze thanks to their advanced technology.

### La commedia dell'arte

"La commedia dell'arte" [65] 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 (Figure 2.17) 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 sensors and, apart from Arlecchino, they totally lack expressivity. Furthermore, the small size was not suitable for being seen from a distance as it could happen on a real stage.

### Robotic Terrarium

One of the other first performances with a robot is the one written by the University of Cambridge [34] called "Robotic Terrarium", which consists of an anemone-shaped robot focused on performing actions based on the interaction with the public through a camera and some sensor. The problem with this robot is that it has a predefined character, and cannot interpret other characters, unlike a real actor.

Figure 2.17: Robot performing "Lazzo of the statue" [65]

### Herb

Herb [66] 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 (Figure 2.18). "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 remote controls the robot and gives commands with the right timing, so the robot is not autonomous. The results were positive, the play was very appreciated by the public from both artistic and technical points of view. It was only criticized because there were body parts not exploited (like the wheels).

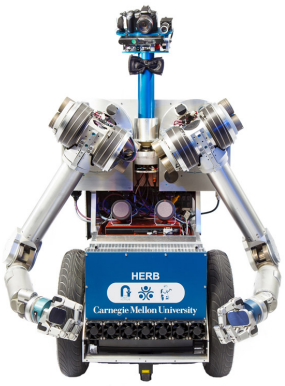



Figure 2.18: Herb [66]

### AUR

AUR [44] is a robot-lamp inspired by the Pixar logo. It can move its arm and change th color and intensity of the light (Figure 2.19). The robot has a script divided into "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 2.19: AUR

### Roboscopie

Roboscopie [56] is a project that involves PR2 (Figure 2.20), 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.

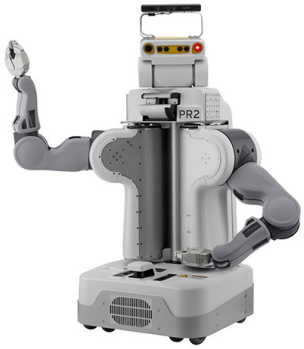



Figure 2.20: PR2

### Nao Stand-Up comedian

NAO was used to make stand-up comedy [41]. In order to allow the robot to perform this type of task, there were several problems to be addressed. From the expressive point of view of the movements, it is difficult to find those suitable for the context, as people are not used to identifying with the limited abilities of the robot. To say its jokes a brand new text-to-speech was used, and the coordination between voice and

movements were done "by hand" because they were not able to find a software able to do this automatically. The robot tells its jokes at the correct timing using software able to determine whether the room is enough silent (the audience is not laughing). The robot is also able to determine if its joke was successful by listening to how loud the laughter is.

### 2.4.2 Improvisation

One of the abilities that an actor has to master is improvisation. There are studies to make a robot improvise [36] [59]. These are quite recent studies and few of these kinds 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 humor of his human companion and then program the robot to improvise actions and speak coherently with its character and the situation.

#### Shimon

Shimon [45] is an interactive robotic marimba player that improvises in real-time while listening to, and building upon, a human pianist's performance (Figure 2.21). 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 2.21: Shimon [45]

## 2.5 Theatrical robots outside the stage

The scope of this kind of robot is not only to entertain the public on stage, but theatre is a test field for features useful in other contexts, for example, in the care of elderly people [64], a sector that will require more and more workers, and robots might assist them in their job. Theatrical robots will be useful in this field because they are not only machines that do actions, but also can entertain the elderly with funny expressions and keep company with them when they feel alone. Another possible application

field is education [32]. At the University of Michigan, researchers developed a project to promote STEAM abilities (Science, Technology, Engineering, Arts, and Mathematics) to the children. The primary objectives of this project were (a) to provide a multi-week after-school program to give young children exposure to social robotics and the arts and (b) to learn how to design better longitudinal child-robot interaction studies conducted outside a controlled laboratory environment. A further objective was to gather data regarding children's preference for and acceptance of different robotic platforms according to the robot's form and appearance. Children developed a play using 8 different robots that they programmed and decorated, supported by instructors. The results of this experiment showed that the children improved their interest in theatre, but especially in robotics, as the instructors wanted to do.

## 2.6 Conclusions

From this chapter, it is clear that there are almost infinite ways to design shows in which robots take part or even are the protagonists because there are many emotions and ways to represent them. The audience can be amazed both thanks to the performance on stage and thanks to the technological level. There are ways to evaluate the software and hardware of robots, but there are no deterministic methods to define the performance on the stage because it is a subjective characteristic. The most important thing is to impress your audience.

Our project wants to propose a fairly technologically advanced robot, widely customizable by different directors and set designers to be able to take part in several different shows.

# Chapter 3

# The Project

We decided to realize a robot able to perform in a theatrical play. The robot is on stage with human actors, it says its lines and shows emotions coherently with the story we want to represent. The robot is autonomous on the stage, although it needs an operator to control the correct flow of the robot's actions (entering the scene) and intervene in case something goes wrong. The robot's animations are predefined and can be selected in the script, the software of the robot is in charge of following it and synchronize all the actions coherently with Walt Disney's principles of animation (see Section 2.2.2). Accordingly to David Lu's classification 2.4, the robot has a hybrid level of automation with closed-loop control, belonging to Class VI and Category 3 (see Section 2.4).

## 3.1 System Requirements

To design a platform suitable to convey emotions and play on stage, we identified the requirements reported below.
**Movement**

1. **Locomotion**: possibility to move from one point to another.
2. **Agility**: the robot should be able to move in narrow spaces.
3. **Localization**: The robot must localize itself on the stage.
4. **Obstacle Avoidance**: The robot must be able to perceive and avoid unexpected obstacles while moving.
5. **Navigation**: The robot should be able to reach any point on the stage with a given precision.
6. **Mobile Parts**: The robot should be able to move parts to have good expression possibilities.

**Shape**:

1. **Shape**: the aspect of the robot and its size should be compatible with what is needed on the scene, and should allow for actor-level performances.
2. **Elastic body**: The robot should keep the same volume while squashing and stretching.

3. **Expressivity**: the robot's emotions must be clearly perceived by the audience even from a distance.

**Sense and Compute**:

1. **Sensors**: the type of sensors available on the platform should enable perception of the scene sufficient to play.
2. **On-board processing**: processing should be performed on-board.
3. **Communication**: it should be possible to exchange information with external devices for synchronization.
4. **Internet**: The robot must not rely on an internet connection, because it can be insufficient or unstable in our environment.
5. **Timing**: The robot should keep the correct timing in the script performing its action at the right moment.
6. **Speaking**: the robot should be able to play the lines in the play giving intentions and emotion to its voice.

**Other requirements**:

1. **Weight** The robot should be easily transportable in different places.
2. **Easy to modify**: The robot's aspect must be easy to modify even between a scene and another.
3. **Modular**: possibility to add, change or delete hardware and software components without any major impact on the performance.
4. **Costs**: the robot should keep the costs as low as possible.

## 3.2 Shape

We decided to build a non-humanoid robot with low movement capabilities to keep the complexity low, because, as shown in the previous chapter, even with a simple shape robots can perform well on stage and result very expressive. For the aspect of the robot, we have taken inspiration from "Minions" (Figure 3.1), funny egg-shaped aliens protagonists of four films of Universal Studios, so the silhouette of the robot is similar to theirs, it has yellow skin and blue trousers with a belt to divide the two parts. Accordingly to the first rule of Walt Disney's principles of animation [46] the robot will be very flexible and made with tensile mechanisms, elastic components and a soft exterior. We have taken inspiration from Blossom, described in Section 2.2.3 [61]. The robot has a tilting hip bone to give the possibility to bow in 4 different ways corresponding to different expressions. The eyes are inspired by snails (which in turn inspired some extra-terrestrial creatures in movies): they are spherical and mounted on flexible pipes, that, depending on how they bend, will show different emotions and can drive the attention in different directions. This eye shape is chosen because it is relatively simple to build, it has ample range of motion and the direction in which the robot looks is easily visible by the public even from a long distance. The robot speaks through two speakers mounted inside the body, but it does not have a mouth, because of the difficulty to perfectly emulate the lip-synch, but also because it is not fundamental to make it seem to speak with the mouth, being the

body movement the main expressive mean on stage. Moreover, there are successful examples of characters speaking without a mouth (e.g., Hello Kitty). The robot does not have arms due to the difficulty to mount believable multiple D.O.F. arms on a flexible body. They would also have led to currently unsustainable expectations that would lead the robot to be less appreciated by the public (Uncanny Valley [52]). The robot, taking inspiration from Herb (Figure 2.18) and PR2 (Figure 2.20) is able to move around on the stage while avoiding unexpected obstacles and reaching specific targets. To show its emotions, the robot will use body and eye movement, but will also have the ability to move its base reflecting its own emotion.

Figure 3.1: Minions

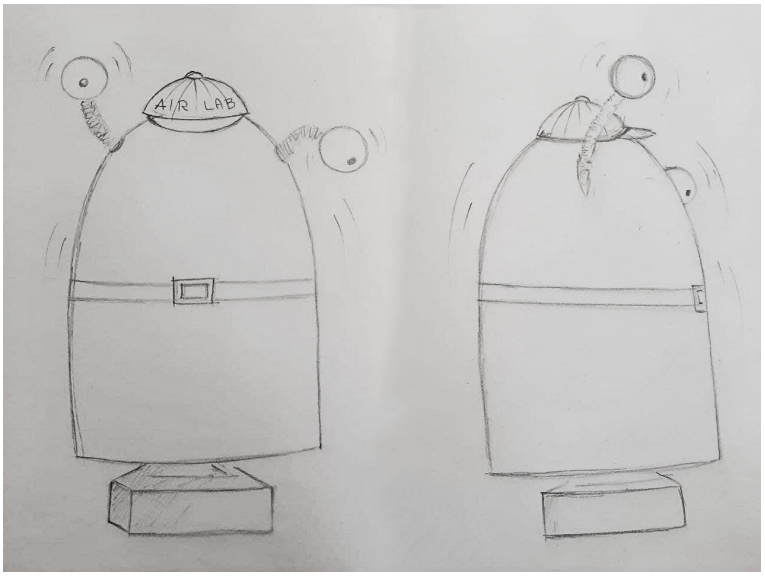



Figure 3.2: First robot sketch

Figure 3.3: The robot on the stage

# Chapter 4

# Platform and Body

## 4.1 TriskarOne

We have chosen the already present robotic platform called: *"TriskarOne"* (Figure 4.1), 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 [33].

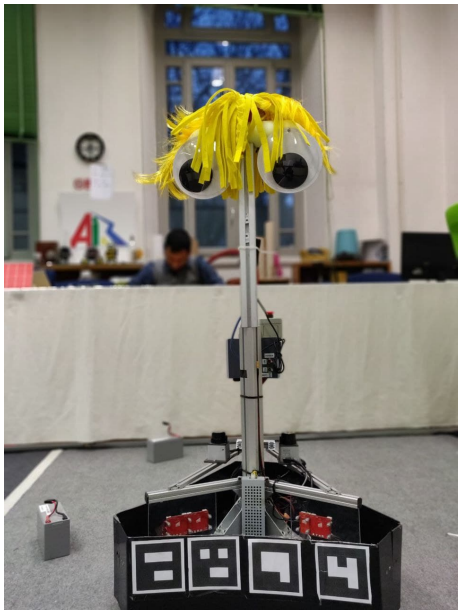


Figure 4.1: TriskarOne before our modifications

### 4.1.1 Structure

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.

### 4.1.2 General Modifications of the structure

To adapt the structure to our purpose and solve some issues encountered, some changes have been made. The screws of the bumpers were changed to make them easier to remove and put back on, so it is easier to access the inside of the base and change the batteries when they are discharged. The place of the box containing the switches was moved at the base of the robot, to make it easier to access when the body is mounted. A box containing the DC-DC power system, all the circuits to supply power to Arduino and servo motors were added. The circuit to check the battery status was redeveloped because the old one contained no longer used parts and was in part damaged. The position of the emergency button was modified to make it more accessible when the body is mounted. The 2 aluminum supports of the laser scanners were reinforced and regulated, because there was a deviation of 3 degrees that caused wrong readings.

## 4.2 Body

An external body was not present and had to be designed to match the specifications, and implemented.

The waist was designed mimicking the kinematic proposed by the much smaller robot Blossom [61]. A structure is suspended with an elastic connection to a central pole and tilted by wires actuated by servomotors. The much bigger and heavier structure required by this robot called for special arrangements for both the elastic suspensions and the actuation. The robot has a tilting hip bone that consists of a circle made with a PVC pipe, light and robust enough. This pipe is hanging and supported by 3 elastic ropes connected to the top of the central pole. Iron cables are connected at 4 points of the PVC circle; they can be rolled up or released by 4 servo motors mounted on the base. In this way, the ring is bowed in different directions. The left and right motors are used to bend the body to the side. The rear motors are meant to make the robot fold forward and backward. This structure is mounted taking into consideration to avoid interference with the laser's field of view. On the base PVC circle aimed at being tilted, a second circle is fixed integrally, at a 50 cm higher position, to hold the eyes mechanisms. The internal body is shown in Figure 4.2. The waist is covered by soft polyurethane foam that brings to a general image of an ovoid, as shown in Figure 4.3.

The foam is 1 cm thick, but inside 6 foam reinforcements 3 cm thick are glued to provide structure. The keystone of this structure is removable to provide easy access to the inside from the top. This foam structure is fixed to the PVC circles by iron wires. On top of the foam there is a Lycra dress, that can easily be changed when needed. The upper part of the dress is egg yolk yellow and simulates the skin, the lower part is blue to simulate a kind of overalls. The two parts are joined by a fake brown belt. To close the top of the robot there is a blue hat made in fabric, which also has the function of hiding the microphone without interfering with sound waves (Figure 4.4 The lower part of the base is simply covered by brown fabric to simulate shoes. In between this and the higher part, a fence is left for the laser beams, described in Section 6.3.

### 4.2.1 Eyes

Each eye is composed of 3 parts: a polystyrene sphere which represents the eyeball, connected to a PVC elastic pipe (electrical conduct) by a plastic screw (the one of

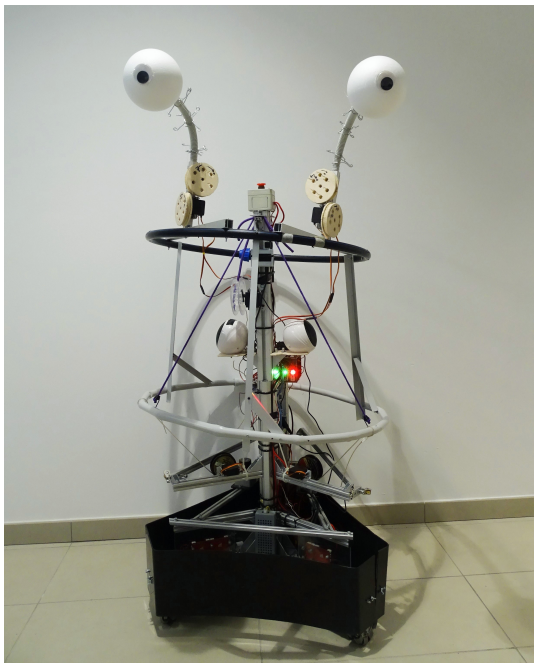


Figure 4.2: Interior of the robot

the common PVC bottles) to make it easy to be removed. On the PVC pipe there are 16 passers-by that keep 4 nylon wires in position at 4 distances from the eye. The wire is quite resistant and can support 14kg, enough for the needs. The wires are rolled up or released on by 2 servo motors mounted at the base of the eye using wooden wheels. Depending on the angle of rotation of each of the two motors the eye can rotate and look in different directions. The field of movement of each eye can be approximated by a spherical cap. The two eyes are mounted on a higher ring structure, mentioned above, jointly liable to the waist to make them move together. A picture of the eye mechanism is reported in Figure 4.5. All mechanisms are covered with the same fabric of the skin of the robot.

Figure 4.3: The foam under the skin of the robot


Figure 4.4: The exterior aspect of the robot

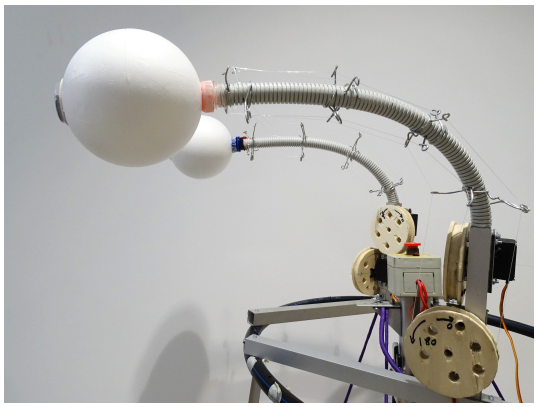


Figure 4.5: Detail of the eyes mechanism

# Chapter 5

# Software Architecture

The robot's software must be able to control all robot movements in parallel. It must also be able to manage localization, obstacle avoidance, and the correct flow of speech. Given the large number of functions to be programmed, a framework for which there are pre-existing libraries was selected. It would also be possible to add more functionalities in the future.

## 5.1 ROS

Much of the following information is taken from "ros_wiki" [5], a website that contains all the information needed to program ROS explained in a simple but comprehensive way.

ROS(Robot Operating System) is an open-source, meta-operating system that provides the services one would expect from 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 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 storage of data on a Parameter Server.

#### Why ROS?

ROS is one of the most used frameworks in robotics. It has a lot of support and there are a lot of 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. The distribution for Ubuntu 20.04 LTS is ROS Noetic [6]. The choice for the operating system was Linux, as typical with autonomous robots.

### 5.1.1 Conceptual Overview

Here, some concepts of ROS that will be used later are introduced [7].

### File system

The file system-level concepts mainly cover ROS resources that one may have on disk, such as:

- **Packages**: Packages are the main unit for organizing software in ROS. A package may contain ROS runtime processes (nodes), a ROS-dependent library, data sets, configuration files, or anything else that is useful to be organized together. Packages are the most atomic build item and release items in ROS. Meaning that the most granular thing that one can build and release is a package.
- **Message (msg) types**: Message descriptions, stored in *my_package/msg/MyMessageType.msg*, define the data structures for messages sent in ROS.
- **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 [8].

### Computation Graph

The Computation Graph is the peer-to-peer network of ROS processes that are processing data together. The basic Computation Graph concepts of ROS are *nodes, Master, Parameter Server, messages, services, and topics*, all of which provide data to the Graph in different ways.

- **Nodes**: Nodes are processes that perform computation. 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.
- **Master**: the ROS Master provides name registration and lookup to the rest of the Computation Graph. Without the Master, nodes would not be able to find each other, exchange messages, or invoke services.
- **Parameter Server**: The Parameter Server allows data to be stored by key in a central location. It is currently part of the Master.
- **Messages**: Nodes communicate with each other by passing messages. A message is simply a data structure, comprising typed fields. Standard primitive types (integer, floating-point, boolean, etc.) are supported, as well as arrays of primitive types. Messages can include arbitrarily nested structures and arrays (much like C *struct*).
- **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,

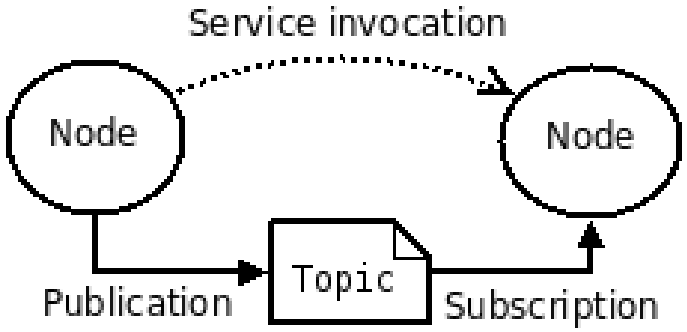



Figure 5.1: Communication between nodes

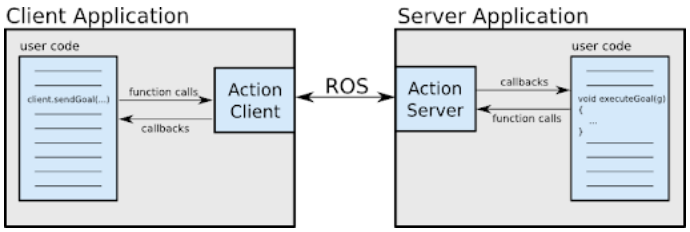



Figure 5.2: Communication in ActionLib

publishers and subscribers are not aware of each others' 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 the right type.

- **Services**: The publish/subscribe model is a very flexible communication paradigm, but its many-to-many, one-way transport is not appropriate for request/reply 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 5.1)

- **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 [8] provides tools to create servers that execute long-running goals that can be preempted. It also provides a client interface in order to send requests to the server. (Figure 5.2, 5.3).

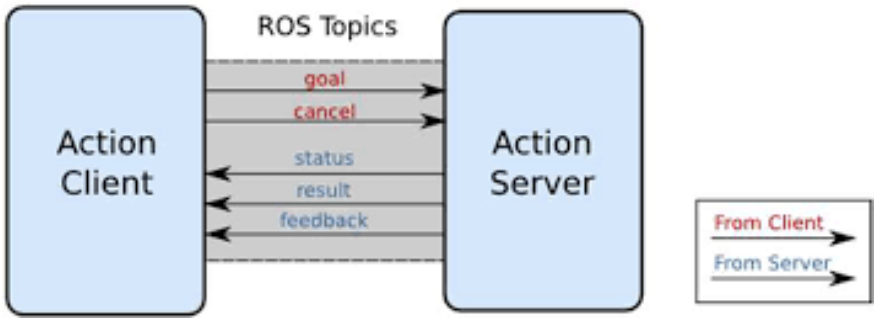



Figure 5.3: ActionLib interface

# Chapter 6

# Sensors and Actuators

## 6.1 Computer

The software runs on a Shuttle DH310 Mini PC [9] which mounts Intel I 7-8700 CPU, 8GB 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 6.1).


Figure 6.1: Shuttle DH310

## 6.2 Arduino

To control the servo-motors and the battery status we used Arduino Uno [10](Figure 6.2). 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 6.2: Arduino Uno

### 6.2.1 Battery

The robot's power supply is composed of 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. Between the two

ends of the battery there are two resistors of $22k\Omega$ and $4.7k\Omega$. Arduino pin A0 is connected as shown in the figure. The voltage divider is necessary because the Arduino pins support a maximum of 5V input. The voltage value at pin A0 is checked by Arduino every 10 seconds. When the batteries have discharged the voltage across the resistors decreases, if this reaches the value of 21V (3.70V at pin A0) the batteries are considered discharged and a buzzer sounds with an interval of 10 seconds. The two resistors have value to have current in the order of mA. The power consumption of this circuit is therefore about 1mWh, which is acceptable considering the nominal energy of the batteries. The schematics are shown in Figure 6.3

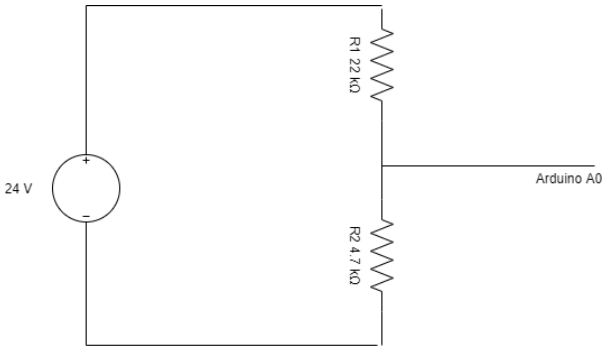



Figure 6.3: Schematics for reading the battery state

### 6.2.2 Eyes and Body

#### Hardware

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 (Figure 6.4 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 6.4: MG996R

## 6.3 Laser Scanners

#### Hardware

The robot mounts 2 *Hokyuo URG-04LX-UG01* laser scanners (Figure 6.5). The light source of the sensor is an infrared laser. The scan area is $240^{o}$ semicircle with a

maximum radius of 4000mm. The Pitch angle is $0.36^{o}$ and each sensor outputs the distance measured at each point (Figure 6.6). Distance measurement is based on the calculation of the phase difference between emitted and detected signals, due to which it is possible to obtain stable measurement with minimum influence from the object's color and reflectance. The sensor is designed for indoor use only. Lasers can have several sources of error. As they rely on light reflection, 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 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 random and unexpected measurements may occur. 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 6.5: Hokuyo URG-04LX-UG01

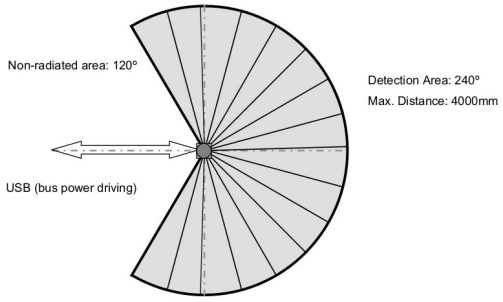



Figure 6.6: Hokuyo's range

## 6.4 Motors

### 6.4.1 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 take any desired orientation. This type of movement requires a specific kind of wheel, like the one that is shown in Figure 6.7. This type of wheel 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 6.7: Holonomic Wheel

This kind of platform requires at least three motors and wheels to move. Figure 6.8 depicts a holonomic platform with three motors. In this configuration each motor is placed every $\frac{2\pi}{3}$ angle on a circular reference and each wheel has a rotation of $\frac{\pi}{2}$ w.r.t the motor's axis

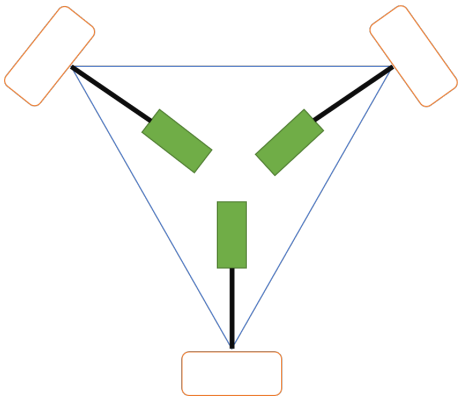


Figure 6.8: Motors configuration

### Motors

The three motors are MAXON 118798 DC motor RE36 GB 70W KL 2WE,whose characteristics are reported in Figure 6.9. 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 their 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 are an index pulse PO which is generated once for each full rotation of the code-wheel.

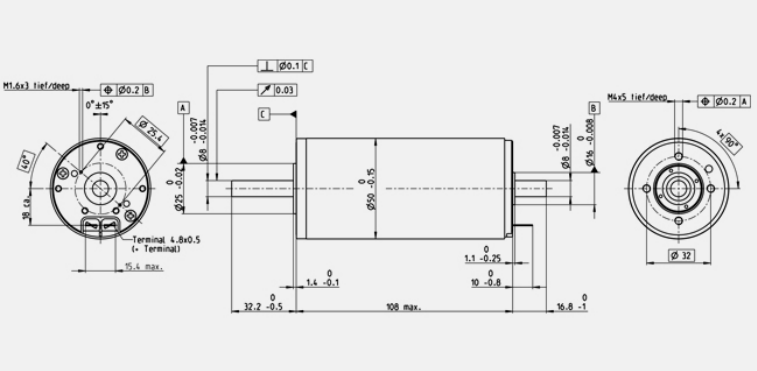


Figure 6.9: MAXON 118798 DC motor characteristics

### 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 controlling different types of motors that can be modeled as a second-order system, where the input is the voltage applied to the motor armature and the output variable is the motor angular speed.

### Kinematics

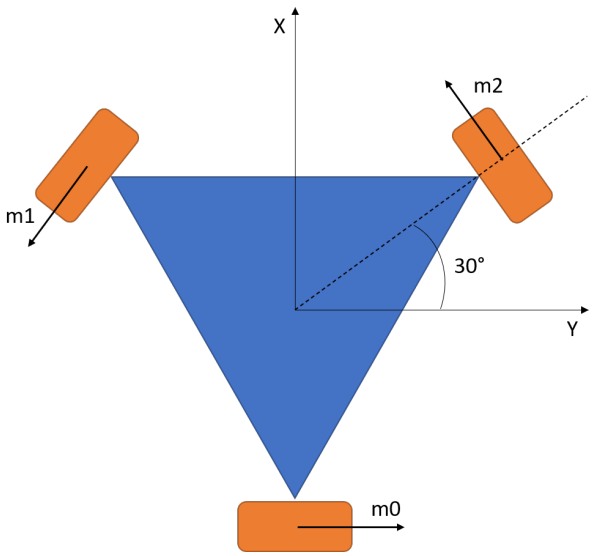



Figure 6.10: Holonomic platform with three motors in a Cartesian plan. m1, m2 and m3 represent the motors

Using the configuration shown in Figure 6.10, it is possible to determine the velocity of each motor $< m_1, m_2, m_3 >$ given the desired velocity triplet $< V_x, V_y, V_\omega >$, where $V_x$ and $V_y$ are the velocities in the x and y axis directions, respectively, and $\omega$ is the rotational velocity with respect to the center of the robot. Then, the system could be described as follows:

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

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:

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

**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 number of pulses counted during the time window and PT is the number of pulses that are necessary to do a whole turn in the wheel, $\Delta t$ is the time elapsed between two consecutive calculations.

## 6.5 Audio

Figure 6.11: Speakers and Microphone

The robot mounts two 2.5 W amplified speakers with aux input, connected to a USB port for power supply. The robot is equipped with an omnidirectional condenser microphone with a maximum range of about 5 meters. To reproduce the sounds it uses 2 2.5W USB-powered speakers. Both devices are connected to the computer via an external USB sound card, which reduces both input and output noise compared to the computer's internal sound card

## 6.6 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.

## 6.7 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 (Figure 6.12 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.

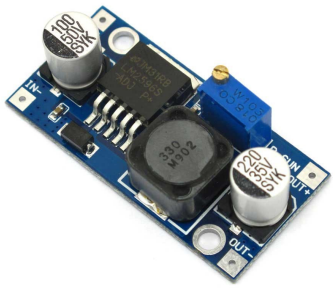



Figure 6.12: DC-DC LM2596

# Chapter 7

# Navigation

The robot should be able to map the environment, localize in it and move in the 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. Much of the following information is taken from "ros_wiki" [5], a website that contains all the information needed to program ROS explained in a simple but comprehensive way.

## 7.1 Prerequisites

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 9.1.2

#### Odometry

The Odometry is calculated by the *odometry_publisher* and data are published on the topic *"odom"* of type *"nav_msgs/Odometry"* [11]. The Node takes as input $< V_x, V_y, V_\omega >$ by subscribing to the topic *vel* and computes the pose $< X, Y, \theta >$ of the robot w.r.t the initial position using reverse kinematics using the formulas below reported.

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

$$\delta x = (V_x * \cos\theta - V_y * \sin\theta) * \delta t$$

$$\delta y = (V_x * \sin\theta + V_y * \cos\theta) * \delta t$$

e:

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

$$X = X + \delta x$$

$$Y = Y + \delta y$$

where $\delta$ is the difference between one time interval passed between one misuration and the precedent one.

**TF**

At an abstract level, a transform tree [12] 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 a long time they could have changed a little their positions.

## 7.2 Robot Mapping

The first thing to get is the map of the environment. To do so we use a ROS library called *"gmapping* [13]. 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 and builds a variable number of possible maps (each one is called a particle). The best particle is selected taking into 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.

### 7.2.1 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 showed good behaviour.

## 7.3 AMCL

Once we have a map the robot must localize itself in it; this is done using the ROS library AMCL [14]. 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.

### 7.3.1 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 [67] 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 the 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*: 18.0 Rotational movement required before performing a filter update.(in $\frac{\pi}{value}$ radians)

Since the robot can move also by a little, a very frequent filter update is necessary, so 5 cm and 10 degrees 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 may 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 from the rotational component of the robot's motion.
- *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 less predictable friction on the ground may cause some issues.

## 7.4 Move Base

The *move_base* [15] 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 (Figure 7.1). 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.

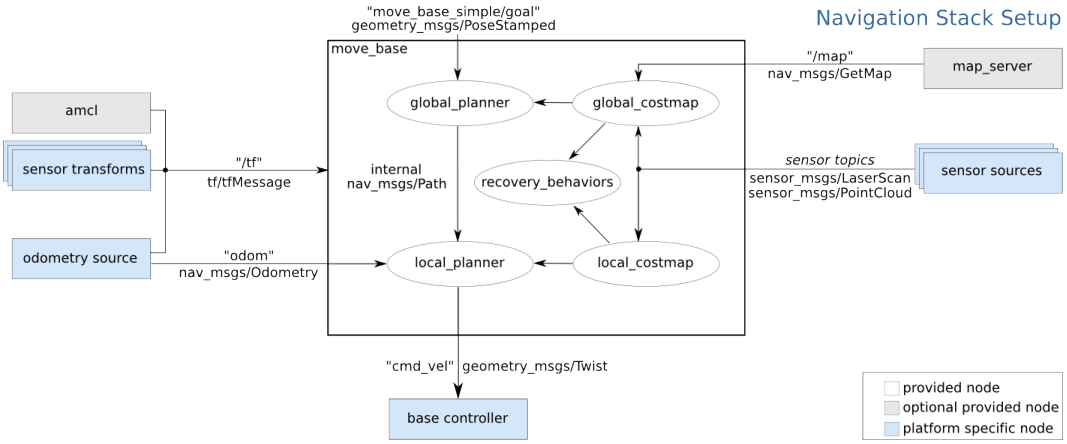



Figure 7.1: Move Base Architecture

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. To properly use *move_base* we have set some parameters.

### 7.4.1 *move_base* general parameters

- planner_patience: 3.0 - How long the planner will wait in seconds in an attempt to find a valid plan before space-clearing operations are performed.

- *controller_patience*: 3.0 - How long the controller will wait in seconds without receiving a valid control before space-clearing operations are performed.

These timers were set taking into consideration that a too short time will cause the planner to always fail in case of a small uncertainty, but a too long one will cause the planner to wait for too long before recovery, causing the robot to be stuck for too long (which is bad in a theatrical play).

### 7.4.2 Global Planner

Global Planner is in charge to plan the path to reach the goal. Parameters should be set for the algorithm; with this selection Global Planner produces a smooth path that it is more suitable for our purpose of mimicking the human movement:

- *use_dijkstra:*true If true, use Dijkstra's algorithm. Otherwise, *A**.

- *use_quadratic*: true.

- *use_grid_path*: false.

The different behaviours for the global planner are reported in Figure 7.5. Each planned path is characterized by a cost that determines its quality. To calculate it there are unlisted parameters: *cost_factor, neutral_cost, lethal_cost*. Besides these parameters, there are other three unlisted parameters that actually determine the quality of the planned global path . They are *cost_factor, neutral_cost, lethal_cost*:

- *neutral_cost*: 50

- *cost_factor*: 0.8

- *lethal_cost*: 253

where:

$$cost = neutral\_cost + cost\_factor * costmapcostvalue$$

Incoming costmap cost values are in the range 0 to 252. With *neutral_cost* of 50, the *cost_factor* needs to be about 0.8 to ensure the input values are spread evenly over the output range, 50 to 253. If *cost_factor* is higher, cost values will have a plateau around obstacles and the planner will then treat (for example) the whole width of a narrow hallway as equally undesirable and thus will not plan paths down the center. For *lethal_cost*, setting it to a low value may result in failure to produce any path, even when a feasible path.

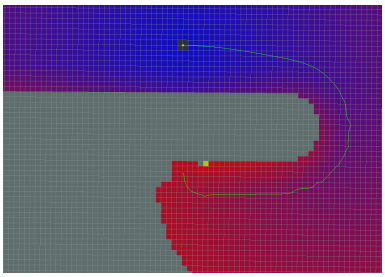


Figure 7.2: global planner standard behaviour

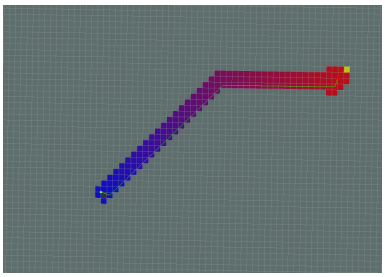


Figure 7.3: use_dijkstra:false

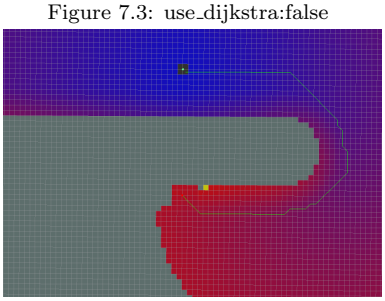


Figure 7.4: use_grid_path:true

Figure 7.5: Global Planner Behaviours

### 7.4.3 Local Planner

Given a global plan to follow and a costmap, the local planner produces velocity commands to be sent to a mobile base. Among the different local planners provided by the ROS navigation stack *dwa_local_planner* [16] is chosen because it provides an easy to understand interface and it is suggested to be used with holonomic robots. The goal of dwa_local_planner is to produce a $(v, \omega)$ pair that represents a circular trajectory that is optimal for the robot's local condition. DWA reaches this goal by searching the velocity space in the next time interval. The velocities in this space are restricted to be admissible, which means that the robot must be able to stop before reaching the closest obstacle on the circular trajectory dictated by these admissible velocities. Also, DWA will only consider velocities within a dynamic window, which is defined to be the set of velocity pairs that are reachable within the next time interval given the current translational and rotational velocities and accelerations. DWA maximizes an objective function that depends on (1) the progress to the target, (2) clearance from obstacles, and (3)forward velocity to produce the optimal velocity pair. The basic idea of the Dynamic Window Approach algorithm is as follows:

1. Discretely sample in the robot's control space (dx,dy,dtheta).

2. For each sampled velocity, perform forward simulation from the robot's current state to predict what would happen if the sampled velocity were applied for some (short) period of time.

3. Evaluate (score) each trajectory resulting from the forward simulation, using a metric that incorporates characteristics such as proximity to obstacles, proximity to the goal, proximity to the global path, and speed. Discard illegal trajectories (those that collide with obstacles).

4. Pick the highest-scoring trajectory and send the associated velocity to the mobile base.

5. Rinse and repeat.

Among the huge number of parameters that can be set, here are reported some of the most significant ones
**Goal Tolerance Parameters**:

- *yaw_goal_tolerance*: 0.2 - The tolerance in radians for the controller in yaw when achieving its goal.

- *xy_goal_tolerance*: 0.50 - The tolerance in meters for the controller in the x and y distance when achieving a goal.

- *latch_xy_goal_tolerance*: true - If goal tolerance is latched, if the robot ever reaches the goal xy location it will simply rotate in place, even if it ends up outside the goal tolerance while it is doing so.

We decided to have a large goal tolerance because it is more important for the robot to achieve the goal as soon as possible rather than to be precise. We are a bit less permissive regarding the yaw tolerance because the robot must look in the correct direction (e.g., the public).
**Forward Simulation Parameters**:

- *sim_time*: 2.5 - The amount of time to forward-simulate trajectories in second.

- *sim_granularity*: 0.025 - The step size, in meters, to take between points on a given trajectory.

- *vx_samples*: 10 - The number of samples to use when exploring the x velocity space.

- *vy_samples*: 10 - The number of samples to use when exploring the y velocity space.

- *vth_samples*: 15 - The number of samples to use when exploring the $\theta$ velocity space.

The parameters represent how many trajectories the planner simulates and for how much time. The more samples you get and the more time you simulate them the more possibilities you have to find the best trajectory to reach the goal, but the more it requires a computational power. Making some experiments these values are found to be the best balance between precision and efficiency.
**Trajectory Scoring Parameters**:

- *path_distance_bias*:32.0 - The weighting for how much the controller should stay close to the path it was given.

- *goal_distance_bias*:24.0 - The weighting for how much the controller should attempt to reach its local goal, also controls speed.
- *occdist_scale*:0.01 - The weighting for how much the controller should attempt to avoid obstacles.
- *forward_point_distance*:0.325 - The distance from the center point of the robot to place an additional scoring point, in meters.
- *stop_time_buffer*:0.2 - The amount of time that the robot must stop before a collision in order for a trajectory to be considered valid in seconds.
- *max_scaling_factor*:0.1 - The maximum factor to scale the robot's footprint by.

As we mentioned above, DWA Local Planner maximizes an objective function to obtain optimal velocity pairs. the value of this objective function relies on three components: progress to goal, clearance from obstacles and forward velocity. In the ROS implementation, the cost of the objective function is calculated with the objective to get the lower cost:

$$\begin{aligned} cost = pathdistancebias * (distance(m) to path from the endpoint of the trajectory) + \\ + goaldistancebias(distance(m) to local goal from the endpoint of the trajectory) + \\ + occdistscale * (maximum obstacle cost along the trajectory in obstacle cost(0 - 254)) \end{aligned} \tag{7.1}$$

The values of these parameters were chosen making some experiments.

### 7.4.4 Costmap Parameters

As mentioned above, costmap parameters tuning is essential for the success of local planners (not only for DWA). In ROS, costmap is composed of static map layer, obstacle map layer and inflation layer. The static map layer directly interprets the given static SLAM map provided to the navigation stack. The obstacle map layer includes 2D obstacles. The inflation layer is where obstacles are inflated to calculate the cost for each 2D costmap cell. Besides, there is a global costmap, as well as a local costmap. Global costmap is generated by inflating the obstacles on the map provided to the navigation stack. The local costmap is generated by inflating obstacles detected by the robot's sensors in real-time.
**Inflation Layer**: The inflation layer consists of cells with costs ranging from 0 to 255. Each cell is either occupied, free of obstacles, or unknown. *Inflation radius* and *cost scaling factor* are the parameters that determine the inflation. inflation radius controls how far away the zero cost point is from. The robot should keep itself as far as possible from the obstacles while not doing so much steep curves. The cost function is computed as follows:

$$\exp\left(-1 * cost\_scaling\_factor * (distance\_from\_obstacle - inscribed\_radius) * 253\right)$$

Based on the decay curve diagram, we want to set these two parameters such that the inflation radius almost covers the corridors, and the decay of cost value is moderate, which means a low value of cost scaling factor. These values were found:

- *inflation_radius*: 0.05
- *cost_scaling_factor*: 5.0

**Local Costmap Resolution**: These parameters define the dimensions of the local costmap, a low dimension could cause the robot to not see the obstacles in time, a too big dimension could require more computational power. A good balance is found in these values:

- *width*:3.5 - The width of the local costmap in meters.
- *height*:3.5 - The height of the local costmap in meters.
- *resolution*:0.05 - The resolution of the local costmap in meters/cell.

### 7.4.5 Recovery Behaviours

The move_base node may optionally perform recovery behaviors when the robot perceives itself as stuck. By default, the move_base node will take the actions shown in Figure 7.6 to attempt to clear out space:

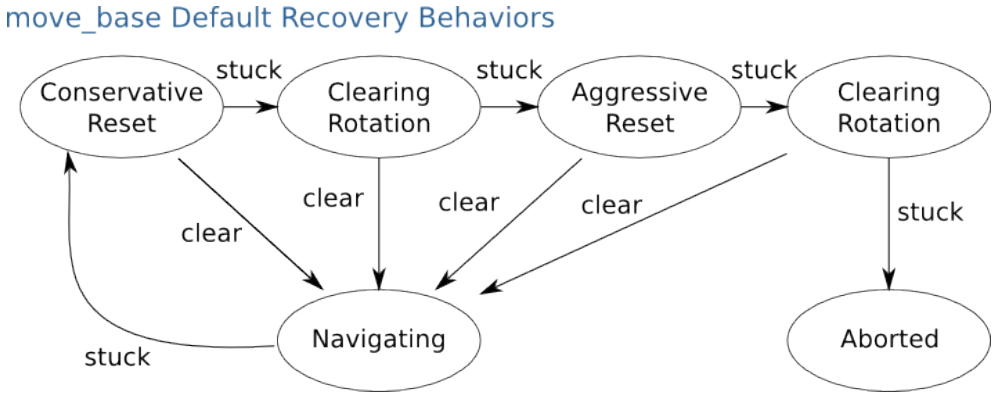



Figure 7.6: Move Base Algorithm

First, obstacles outside a user-specified region will be cleared from the robot's map. Next, if possible, the robot will perform an in-place rotation to clear out space. If this too fails, the robot will more aggressively clear its map, removing all obstacles outside the rectangular region in which it can rotate in place. This will be followed by another in-place rotation. If all this fails, the robot will consider its goal infeasible and notifies the user that it has aborted.

# Chapter 8

# Audio

## 8.1 Speak

The robot must say some lines in the play, so it needs a way to speak. We want the robot to speak in Italian, because we want it to perform for Italian people. The first idea was to use a general-purpose text-to-speech. The Python library *pyttsx3* [17] was tested, but the results were quite bad. In fact, the words it can saying are very difficult to recognize, especially when speaking Italian. The other problem of this library and the other general-purpose text-to-speech tools is that it does not act or express emotion, resulting in a "robotic" voice, not always suitable for our purpose. So we searched for an emotional text-to-speech. We discarded all the software that relies on the internet connection, because any interruption of the signal may cause the robot to stop (and we have some experience with this). We have found some packages suitable for our purpose, i.e., Sonantic [18] and Readspeaker [19]. They perform quite well in reproducing emotional voices, but acting is a different thing. In fact, they have a limited number of emotions that can be selected; moreover, they result quite artificial, in fact when an emotion is selected they speak all the time with the same tone, but in the real world a person can change its tone, the speed and the other characteristics of the voice possibly at each word. We think that general text-to-speech packages present on the market are not enough advanced to perform well in a theatrical contest and none have obtained satisfying results in this field yet. Building our text-to-speech package was a discarded option, because it would be too time-consuming and would require resources exceeding what is possible in a master thesis. Finally, we came up with the idea to dub the robot with a real actor. Although it is not a good solution in terms of the automation of the robot, it is the best solution to make the robot more engaging. We called an actor and recorded all the lines the robot should say, indicating to the actor what emotions he had to express empathizing with the robot. As you can listen, the communication is very effective, there are several changes in pitch and speed. We also choose an actor with a vocal timbre suitable for playing a childlike character like the one that the robot had to play in our trial.

## 8.2 Listen

We want the robot to perform not only monologues, but also dialogues with other actors on stage. We want the robot to listen to the other actor and select the correct timing for doing its action on the script. Firstly we describe what are the characteristics and possible issues that may occur when a human actor speaks.

1. **Errors**: The actors may not remember perfectly their lines, so they may change the words they have to say coherently with the meaning of their line, or they can cut off some forgotten words.

2. **Timing**: The speed of the dialogue may change at any time.

3. **Pauses**: While acting it might happen that the actor does a long pause between a phrase and another, but even between a single word and another. This may be done to create suspense and get attention from the public.

4. **Improvisation**: When an error occurs, often actors try to repair with improvisation, each one listening to the others and trying to move on with the dialogue.

5. **Noise**: The spectators are often not very noisy when actors are speaking, but they may applaud, cough, laugh, and in general produce unpredictable noise. Moreover, there may be background noise, music, and sudden noise caused by the actors moving.

6. **Loud voice**: Actors speak louder than normal, because they must be heard even from the last rows of the theatre. Microphones are rarely used, especially in small theatres.

To select the correct timing while listening to the actor we came up with three solutions:

1. Wait for the time necessary for the actor to say its lines, this time is the average time measured during the rehearsals.

2. Use a speech recognizer to detect the actor's worlds and keep track of the dialogue on the script.

3. Detect if the actor is speaking or not basing on the microphone input.

The first solution was discarded because it encounters problems with issues number 1 and 2: a small change in the lines said by the actor may cause the robot to speak too soon or too late.

The second solution has problems with issue number 1; in fact, even a small change in the lines or an unrecognized word may cause a fault. Even by using only a percentage of correctly recognized words, the robot would encounter faults when the lines are too short or even formed by one word when the actor uses synonyms. For example, in Italian "chiaro" and "capito" have the same meaning in verbal interaction, but they are totally different words. We explored also the possibility to recognize the meaning of the phrase, avoiding the problems of synonyms and shorter or longer sentences. This possibility was discarded because it may cause faults when an actor completely forgets a phrase or a line and tries to improvise changing very hardly the meaning of the phrase. Tuning a fault-tolerant software is also very hard and time-consuming, already trained tolerant software uses the internet connection (e.g., Google [20]), which, as already said, is not stable in a theatrical environment, and requires a good audio signal, so it may not provide perfect performance.

The third solution is the one we have chosen. The actor speaks loudly and, with the correct tuning, the signal can be distinguished from the background noise. It works if the actor says a wrong word or line and if he or she changes the timing. Noise coming from the public can be detected, but if the robot stops speaking or waits while

the public laughs and applauds are correct behavior in the context. The only problem is issue number 3: pauses. In fact, a long pause could be interpreted as the end of the line. To handle this problem, we decided to combine this solution with solution number 1. The robot waits a certain amount of time (tuned during rehearsals) to let the actor speak and making pauses and then starts to listen to detect the end of the line. The only constraint is not to do a pause when the line is about to end. This constraint is considered acceptable and may be managed by inserting a pause before the next line of the robot. The algorithm for speaking detection is reported here below.

**input:** The time **t** to wait before listening to microphone data
**Result:** Check if an actor is speaking or not, terminate when it is not

**while** $timePassed < t$ **do**
| doNothing;
**end**
start to get microphone data
$isSpeaking \leftarrow False$
**while** *not isSpeaking* **do**
| take a chunk of data;
| compute mean square root of the chunk;
| $counterSpeaking \leftarrow 0$;
| $DbValue = 20 * \log(rms)$;
| **if** $DbValue > Threshold$ **then**
| | $counterSpeaking \leftarrow counterSpeaking + 1$;
| **else**
| | $counterSpeaking \leftarrow 0$;
| **end**
| **if** $counterSpeaking = ConsecutiveNonSpeakingChunks$ **then**
| | $isSpeaking \leftarrow True$;
| **end**
**end**

**Algorithm 1:** Speaking Detection Algorithm

The microphone is set to a certain sampling rate (in our case 44kHz), so every 0.023 ms the microphone saves in a buffer the amplitude value of the sound it is sampling. The buffer size (called chunk) is settable and in our case it is set to 4KB. When the buffer is completely filled it is emptied and its contents saved in a data array which will then be processed. A chunk, therefore, contains data for about 0.1 seconds. *Threshold* is the threshold value for the actor's voice loudness, and *ConsecutiveNonSpeakingChuncks* are two parameters that can be set with the ROS Parameter Server. The default values, which have been tuned by making some experiments, are respectively 60 and 7, but they depend a lot on the context in which the robot is located.

# Chapter 9

# ROS Nodes

In this chapter is reported the information about each component in the ROS Code. The meaning of the figures is understandable thanks to the legend reported here below. Some node names have been changed to allow for greater readability.

**Legend**

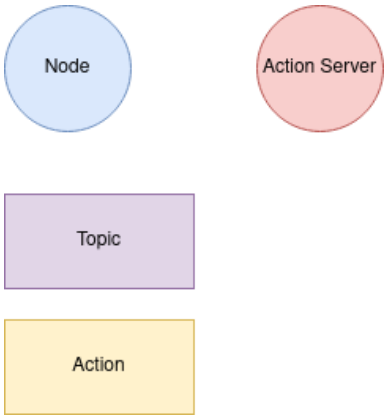



Figure 9.1: Legend for the figures in this chapter

## 9.1 Serial Nodes

Arduino and the Nova Core boards run a ROS node and publish and subscribe to the topics. In this section, we report how these nodes work. Note that it is not necessary to start them with the usual *rosrun* command, but they start as soon as the respective microcontrollers are started. To set up the communication with the master running on the Shuttle we have to run the *serial_node* of the package *rosserial_python* [21]. On the *rqt_graph* the two nodes are called *Serial_Node* followed by a number, but for simplicity we will call them *Arduino Node* and *Nova Core*

### 9.1.1 Nova Core

The *Nova Core* node (Figure 9.2) is in charge of controlling the velocities of the 3 motors. It subscribes to the topic *cmd_vel* of type *geometry_msgs/Twist* [22] where is contained the information on the desired velocities $< V_x, V_y, \omega >$. 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 $V_x, V_y, \omega$ and publishes this information on the topic *vel*.

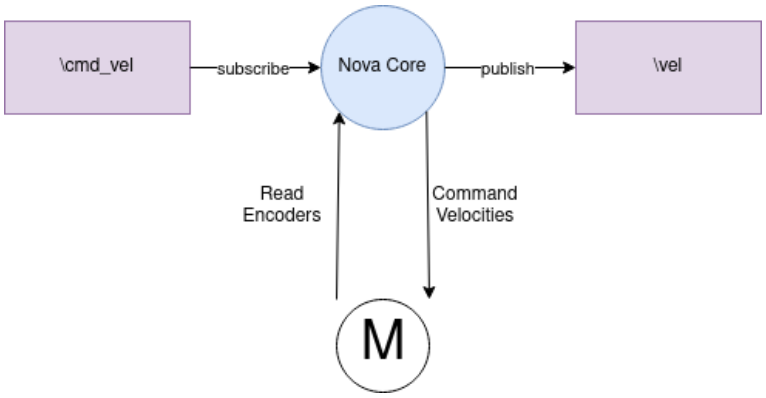



Figure 9.2: Nova Core node and published/subscribed topics

### 9.1.2 Lasers

Two *urg_node* [23] 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°.

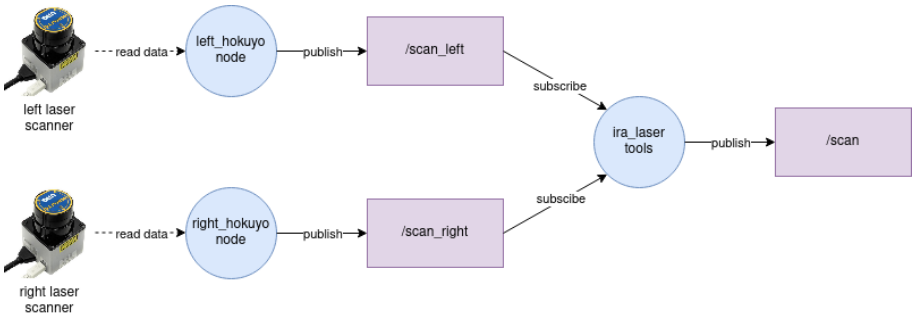



Figure 9.3: Urg Nodes and published/subscribed topics

### 9.1.3 Arduino

The Arduino 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 and reported in Figure 6.3, 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* [24] 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* [25]. We have used the ROS library: *rosserial_arduino* [26], 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 Figure 9.4.

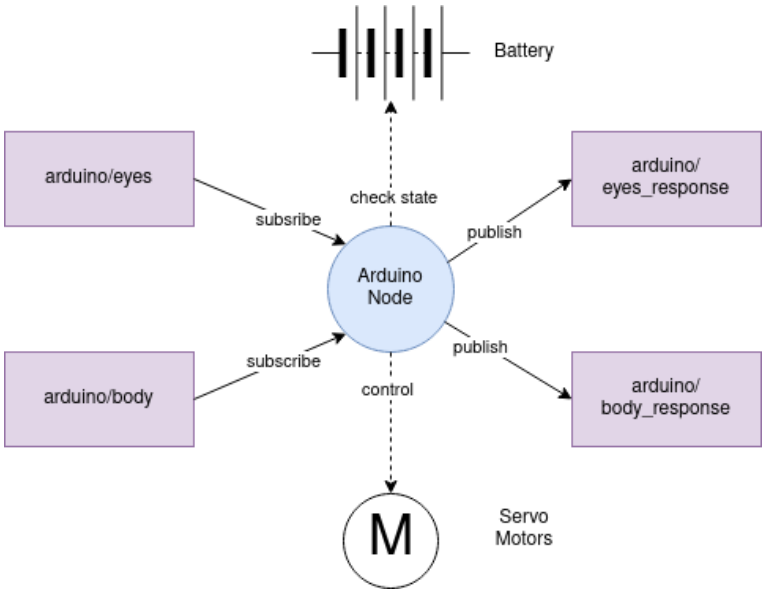



Figure 9.4: Arduino node and published/subscribed topics

## 9.2 Joystick

### 9.2.1 Joy

ROS provides a package called *"joy"* [27] 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.

### 9.2.2 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"* [25]. When is required, the operator can decide to let the robot retry to use move_base by pressing *Retry Move_Base Button* or skip to the next section by pressing *Skip Move_Base Button*, then the joystick_node publishes "1" or "2" on the *"move_base_recovery"* topic of type *std_msgs/Int8MultiArray* [24].

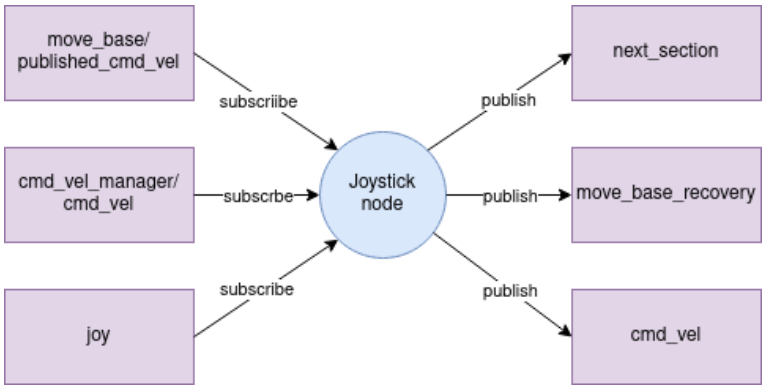



Figure 9.5: Joystick Node and published/subscribed topics

## 9.3 Managers

In this section, the nodes who are in charge of controlling eyes, body and audio are described. All these nodes implement action servers [8]. We decided to use actions because it is suitable for the idea of parallelism: when an action server is called it will perform its assigned actions while the caller can continue to execute its instructions. Actions can be also easily interrupted in case of emergency.

### 9.3.1 Eyes Manager and Body Manager

They are in charge of moving eyes (Figure 9.6) and body (Figure 9.7). 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*: contains the array of positions and speed written on the script.
- *current_movement*: contains the feedback of what movement the robot is performing (the number that defines it).

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 movements, this node is responsible of making the required pause.

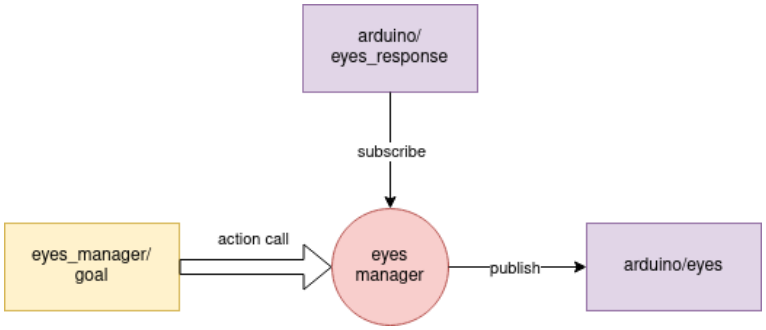



Figure 9.6: Eyes Manager action server and published/subscribed topics

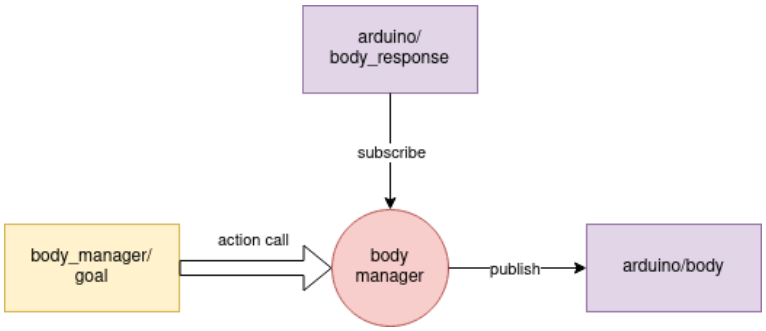



Figure 9.7: Body Manager action server and published/subscribed topics

### 9.3.2 Audio Player

Implements an action server in charge of controlling the speakers (Figure 9.8 The structure of the action is the following:

- *goal*: contains the name of the wav file to play.
- *response*: true if it has finished reproducing the audio file, false otherwise.
- *hasFinished*: is the feedback of the action, false if the robot is still playing the audio, true otherwise.

To reproduce the audio file we have used the library pyaudio [28].


Figure 9.8: Audio Player action server and published/subscribed topics

### 9.3.3 Speech Monitor

Implements an action server that is in charge of listening to the other actors' voices (Figure 9.9) implementing the algorithm described in section 8.2. The structure of the action is the following:

- *wait_time*: the time to wait before starting to listen to the actor.
- *response* true if it has finished listening, false otherwise.
- *hasFinished* is the feedback of the action, false if the actor is still considered to be speaking, true otherwise.

To get the microphone data we used the library pyaudio [28]

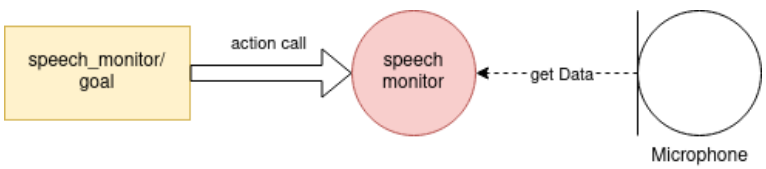



Figure 9.9: Speech Monitor action server and published/subscribed topics

### 9.3.4 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 six numbers that define it).

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 $\theta$ (in radians). The second three numbers represent the velocities on $V_x, V_y, \omega$, 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.

## 9.4 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 subscribes to topics:

- *next_section*
- *move_base_recovery*

It behaves as an action client for the servers:

- *speech_monitor*
- *audio_player*
- *move_base*
- *eyes_manager*
- *body_manager*
- *cmd_vel_maager*

After initialization it runs the main loop. First of all, 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 the main controller calls all the nodes in charge of performing the requested actions passing the necessary information. Finally, 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 break without any action (if required). Finally, if "move_base" is 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.

### 9.4.1 Parameters

Two modifiable parameters can be set:

- *script_path*: the path to the script we want to perform.
- *section_t_start* the section from which to start to read the script (default 1).

The first 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). The second 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.

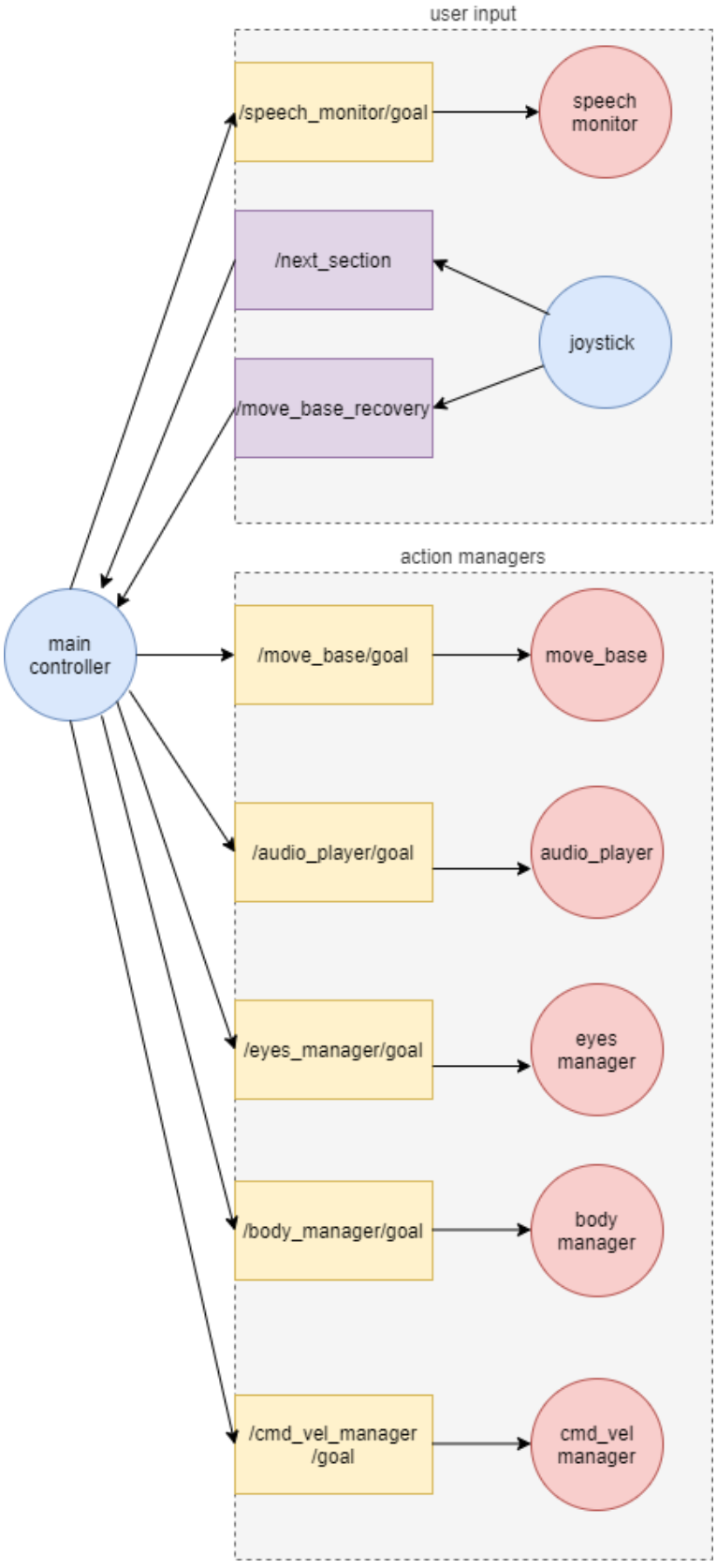



Figure 9.10: Main Controller and published/subscribed topics

# Chapter 10

# Operator Interface

This chapter illustrates how the operator must act to prepare the robot for the show and control it during the performance

## 10.1 Before the play

### 10.1.1 Mapping

The first thing to do is to choose a venue for the performance. It is good that the robot acts on a stage that is not too wide, since the robot, to be able to locate itself, must always be no more than 4m away from a known element. To help the robot in localization, we can place props that are not too jagged and dark and that will never be moved during play. The floor must not be too slippery, as it would lead to mistakes in odometry due to slipping under braking. This is because the lasers have a maximum range of 4 meters, so too far from the walls, it would get lost. Furthermore, too jagged or dark walls would introduce errors caused by unexpected or missing laser beam reflections. Once a suitable place has been chosen, or devised, the operator must create a map of the place and save it in a folder.

### 10.1.2 Rehearsals

Once the stage map has been created, the sections begin to be written. To give the initial position of the robot and try the various positions on the stage we rely on the rviz [29] software.

Once the robot has been given its position on the map, it is possible, using the rviz command, to move around the map and save the desired positions. To move the robot directly without using the planner (for short movements necessary for the scene) it is possible to directly control the robot base using the *manual_move* tag explained in the script. This option greatly speeds up the robot's reactivity and its movement possibilities, but removes the obstacle avoidance. it is therefore good to check that the robot cannot be found on unwanted trajectories. The other movements can be introduced in the script using relative tags. It is possible to try only some sections of the script by modifying the *section_to_start* parameter of the main controller, so as to start the robot from the desired section

## 10.2 During the Play

The robot, once defined what it has to do in the script, is autonomous, but it needs an external operator to carry out the following operations.

- The operator must take care of turning on the robot and starting all the necessary commands. It is also advisable to check that all moving parts are in working order before the performance.
- The operator must take care of locating the robot by providing it with the initial position in which it is located.
- The operator must take care of changing the batteries if the show is very long.
- The operator must enable the robot base to move; it was in fact decided, for safety reasons, that the robot can move the base if and only if the operator is pressing a button on the joystick, this to prevent accidents in case the robot makes mistakes in localization and obstacle avoidance.
- It may happen that the robot cannot reach the position on the stage (if something, for example, completely obstructs the way). The operator must then help the robot to reach the position through the manual controls.
- If required by the script, the operator must advance the sections.

The operator can check the progress of the script through the messages published by the robot on the terminal that show the current status of the robot and the actions it is doing.

## 10.3 Joystick Commands

We need to remote control the robot and be able to intervene when it is necessary. The robot can be controlled by the in the figure 10.1, we decide to use a controller rather than a keyboard because it is easier to control the robot velocity. The commands are:

4) Increase linear velocity.
3) Decrease linear velocity.
1) Increase angular velocity.
2) Decrease angular velocity.
6) Enable manual movement.
5) Enable autonomous movement.
Up arrow) Retry move_base.
Down arrow) skip move_base.
Right arrow) next section.
Right analog stick) move angular.
Left analog stick) move linear.


Figure 10.1: Logitech joystick

# Chapter 11

# Script

The script is a JSON file containing all the actions the robot must do. It is divided into 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"*
**trigger_data**: *trigger_data*
**actions**:{
**move_eyes**:*move_eyes_data*
**move_body**:*move_body_data*
**move_base**: *move_base_data*
**manual_movement**: *manual_movement_data*
**speak**: *speak_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)
- *after_speech*: the section starts after another actor has finished to speak.

**trigger_data** is an integer that is only checked if trigger_type is "after_actor" and it represents the time to wait before listening, see section 8.2 for details.
**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:

1. Standard position.
2. Look right.
3. Look left.
4. Look up.

5. Look down.
6. Cross eyes.
7. 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:

1. Standard position.
2. Bow front.
3. Bow right.
4. Bow left.
5. 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 $< X, Y; \theta >$ direction, the second 3 numbers indicate the velocities in $< V_x, V_y, V_\theta >$ and the last number indicates the amount of time in seconds to wait between these movements.
**speak_data** is a string that contains the audio file to play.
**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** when actions contain the "end" tag the performance is finished. The robot has to start all the actions to do in this state.

# Chapter 12

# Emotion Expression

Let's focus now on how the robot displays its emotions.

## 12.1 Emotion Selection

In chapter 2 we listed the theories that identify what human emotions are and how they are expressed. What emerges is that human emotions are multiple and there are several theories that list them differently. So it is not possible to uniquely determine what and how many emotions are. If we define a list of emotions and their possible definition we need to focus on how to identify them from the text. This is the second source of uncertainty, in fact, although it is possible to broadly identify the main mood of the character in the scene, it can often be influenced by secondary emotions determined by the context, by his relationship with the characters, by his character and even by the author's life. Humans themselves are not able to uniquely identify the emotions of the characters from the text, but often find different shades depending on their interpretation, often also influenced by the thoughts and experiences of the reader. For example, a homophobic person can see a homosexual character more negatively than a more open-minded person, this would lead him to identify negative emotions in the character that would not be noticed by others.

## 12.2 Emotion Expression

How emotions are expressed is itself an uncertainty. There is no unique way to do this, but the expression depends on the context, the character of the person experiencing it, the character of the person in front of him and other factors. For example, people react to tension differently, some cry, others tremble, others even laugh. Therefore, in theatre the Stanislavsky's method is often used, the actor's expressions are not decided a priori, but actors are allowed to resort to the memories of their own life in order to better identify themselves with the character they have to interpret and thus be able to act with the expression that best suit the situation. In the case of the robot, it is not possible to give it memories, but we must resort to those of its programmer (or director) in order to show emotions. Also for scenic reasons it is good that the robot expresses the same feeling in a different way during the show, in order not to be repetitive and "robotic". For these reasons this robot has no software to compute emotions, which would limit its expressive possibilities, but the director is left with full freedom to decide which expressions the robot should make in various situations. On the other hand, the director is not left alone in deciding which expressions to make,

but is helped by suggestions that propose what movements to do and in particular their characteristic speed and acceleration according to the emotions of the robot.

### 12.2.1 Laban Effort Features

The Laban Effort System was used to study how robots can express emotions [48] and comes to our aid. We can therefore define some main features of the movements to perform.

#### 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), and finally whether or not to say a line. The choice of which parts to move is dictated by the director and the stage needs.

#### Shape

How the robot moves is affected by its construction, so apart from the provided movements it is impossible to introduce others unless further moving parts are introduced

#### Space

How the movements are connected with the environment is very important. Since it is possible to rehearse the show before staging, it is possible to determine a priori where the actors and the props will be positioned. A good interpretation of the robot is therefore linked to how he interacts with them. 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

The effort is actually the robot's interpretation of his line 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; in this case we may the limitations due to the robot possibilities.

### 12.2.2 Russel's Circumplex Model

We can also use Russel's Circumplex Model (chapter 2) to define the correct movements. The idea is that an emotion located in the positive part of the arousal axis requires very fast movements, on the contrary one that is in the negative part requires very slow movements. As for the valence, a person who is in a negative situation tends to hide to try to protect himself, while a positive situation leads him to expand. The movements must therefore be aimed at decreasing or increasing the volume of the robot. This thing may not always be true. For example happiness can be expressed differently depending on the character of the person and the intensity of the emotion, anger is a negative feeling, but it requires you to expand the volume by showing strength.

### 12.2.3 Examples of emotional movements

Basing on the theories expressed above, we now list some proposals for 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.

| | **Eyes** | **Body** | **Base Linear** | **Base Angular** |
|---|---|---|---|---|
| **Anger** | up-down<br>fast | bow-bowback<br>fast | forward<br>fast | rotate<br>fast |
| **Sadness** | down<br>slow | bow<br>slow | backward<br>slow | rotate<br>slow |
| **Fear** | down-cross<br>fast | bow<br>fast | backward<br>fast | rotate<br>fast |
| **Embarass.** | down<br>slow | right-left<br>slow | backward<br>slow | rotate<br>slow |
| **Happiness** | up<br>fast | right-left<br>fast | none<br>none | rotate<br>medium |
| **Courage** | up<br>fast | bowback<br>medium | none<br>none | none<br>none |
| **Curiosity** | none<br>none | bow<br>slow | forward<br>slow | none<br>none |
| **Surprise** | devide<br>fast | bowback<br>fast | backward<br>fast | none<br>none |

Table 12.1: Possible emotional movements

#### Explanation

The movements proposed in the table above are only suggestions for the director who can change them at will. Below is an explanation of the meaning of each movement.

- *Anger*: Anger has high arousal and middle valence. The movements must therefore be very fast and the robot must show some kind of power by expanding its body.
- *Sadness*: Sadness is a very negative feeling and has very low arousal. So the robot has to move slowly and bow down simulating crying.

- *Fear*: Fear also has a negative valence, but high arousal., The movements, therefore, indicate a situation in which the robot wants to hide very fast and escape from danger.
- *Embarrassment*: Embarrassment also indicates a certain weakness in not knowing how to answer. Speed is as slow as sadness, but the body moves left and right to make tenderness.
- *Happiness*: Happiness has high valence, but arousal depends a lot on the intensity of the emotion, so the speed can be regulated depending on the situation. The robot rejoices by moving its eyes up and down and rotating on itself.
- *Courage*: Courage has high arousal and medium valence. The robot is agitated so it moves quickly and raises its eyes and torso as a sign of strength.
- *Curiosity*: Curiosity has medium valence and arousal. The robot approaches what intrigues him.
- *Surprise*: Surprise is an unexpected situation, it has high arousal so the speed is fast. The valence depends if the surprise is positive or negative. The robot divides its eyes making a movement as absurd as the situation in which it finds itself.

## 12.3 Non-emotional movements

Some robot movements do not express any emotion but are neutral. These movements are necessary for scenic needs and can help make the show more interesting. The localization is necessary in order to position the robot at the correct points on the stage in such a way as to bring out a certain harmony of the scene. The movement of the eyes and body can be used to bring the attention of the robot (and therefore the audience) to a certain point in the scene. They can also be used to make him take actions that support his speech. For example, nod or shake the eyes if the robot says "yes" or "no", or move the body when the robot is speaking as if gesticulating

## 12.4 Conclusions

Given the (almost) infinite possibilities of movement and expressions it is impossible to define what are the correct ones to best interpret his character. So let's leave it to the director to choose what to make the robot do, leaving him/her ample room for maneuver. In chapter 14, we propose an example of the application of the capabilities of the robot that plays a character in a play.

# Chapter 13

# Testing

Before trying to perform a theatrical play, every part has been tested individually. Thanks to the action server pattern, every sensor or actuator is completely autonomous.

## 13.1 Move Base

The first nodes we tested were the ones related to navigation. We started from the launch files already present and combined them with the ones controlling an autonomous wheelchair already developed in AIRLab. The laboratory was used to adjust the "move_base" parameters. In fact, it is an excellent test field. It has a wooden floor (similar to that of a stage) and is quite large, allowing us to simulate the space between the stage and the audience. To make it more uniform, a fabric wall and two cardboard wings were mounted to simulate a stage. Unlike a real stage, AIRLab has several irregularities on the walls making it more difficult to adjust the parameters, but we think it is a good way to test the robot even in the most extreme cases.

### 13.1.1 Mapping

Firstly we have built the map. After some trials we came up with a good tuning of the parameters and we produced the map. Then we modified it using an image editor to remove irrelevant parts of the map. We came up with the map reported in Figure 13.1. Black represents walls, white represents free space, gray represents unknown space.

### 13.1.2 Amcl

Once the map is built we have started to tune the AMCL parameters to localize the robot in it. The experiments were done by moving the robot along a predefined path, or randomly and check if the robot was still localized at the end of the path, and how big was the spread of particles. The robot performs localization quite well, even if sometimes it might have problems in very symmetrical spaces and jagged walls, which is a common problem with this type of localization. In Figure 13.2 are reported an example of good localization of the robot, the position is coherent with the laser scans and the particle cloud is very narrow.

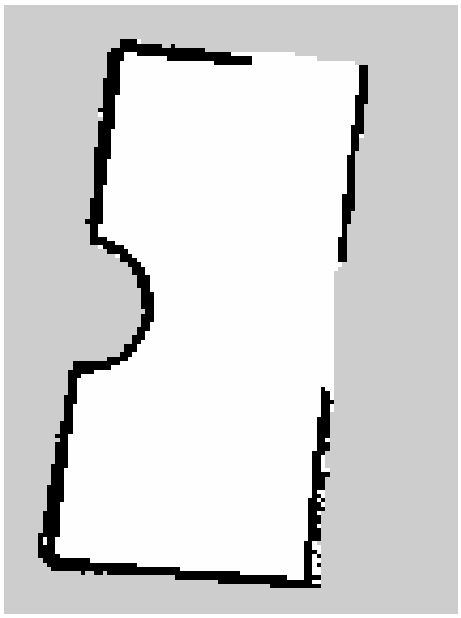


Figure 13.1: One of the maps of AIRLab used to test navigation stack

### 13.1.3 Move Base

After the testing on localization we tested the path planning. Firstly we gave the robot some goals in a clear space and checked if it could reach them without staying still for computation. Then, we put some obstacles on the map and check if the robot could avoid them and reach the goal. Finally, we have tried to give the robot illegal positions and see if it recognizes the problem and communicates it. After some trials we came up with a good tuning of the parameters. The robot still has problems with very narrow spaces, especially the ones caused by unexpected obstacles, which is a common issue with navigation. For this reason we gave the robot the possibility to communicate the problem to the operator who is in charge of unblocking the robot.

## 13.2 Eyes and Body

The testing of eyes and body was done mainly for regulating the servo angles for performing the various movements. It also had the scope to test the durability of the mechanisms.

## 13.3 Microphone

We tested the capability of the microphone to detect whether a person is speaking. We have tuned the parameter to improve the performance, which is quite good in a silent environment, although it needs the actor to speak very loudly in a noisy environment like the AIRLab to be able to distinguish speech from the background noise.

## 13.4 Synchronization

Once we had tested and tuned all the sensors and actuators, we started to check if all the components were correctly synchronized. To do so we have programmed

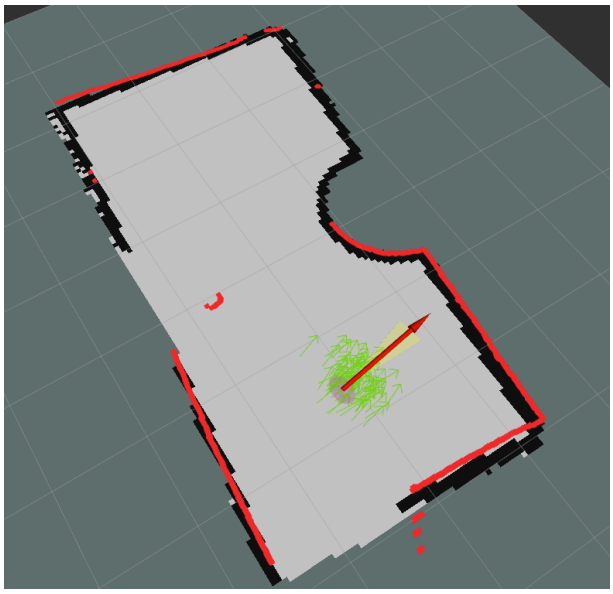


Figure 13.2: Robot localization in the map: the red arrow is the pose of the robot, the green arrows are the particles, the red lines are the detected walls

multiple scripts to simulate all the possible requests that can occur during the play and check whether the robot correctly performs all the requests and, in case of an error, correctly communicates the problem to the operator. The tests were successful although they had to be integrated with real performances because some possible problems and requests might have not been considered.

## 13.5 Play

Finally, a video was shot showing all the acting skills of the robot. More details and results in the next chapter

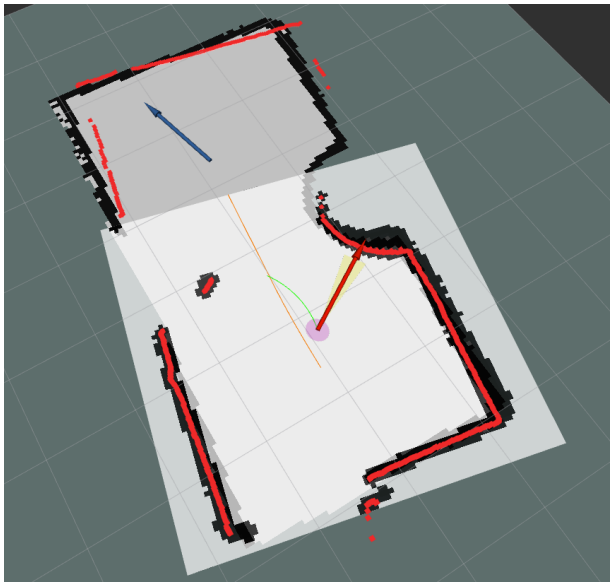


Figure 13.3: Robot path planning, the blue arrow is the final pose, the orange line is the global path, the green line is the local path. The grey spot is an obstacle.

# Chapter 14

# The Play

In this chapter we show an example of using the robot and how it was decided to express its emotions thanks to its possibilities of movement

## 14.1 Pinocchio

We want to represent a small opera with these characteristics:

- It should be for children since we have developed a cartoon-like robot.
- The robot should act like a robot, not a human to result more believable and exploit its true nature.
- Not the whole opera will be represented, the chosen small part should contain many different emotion expression possibilities for the robot.

We have chosen a scene of "Pinocchio" [37] that matches these requirements. The protagonist is a puppet who wants to become a child. We have chosen to rewrite the opera to make Pinocchio a robot who wants to be a child and Geppetto the engineer who has built it. The part of "Pinocchio" that we have chosen is when Pinocchio is eaten by the big shark and in its stomach he finds Geppetto. In this section, Pinocchio feels fear because it is chased by the shark, sadness when he finds out that he will spend the rest of its life in the stomach of the big fish and happiness when it finds his father.

## 14.2 Sections

Section by section illustrates the emotions that the robot is supposed to show, the choices of movements and the lines that the robot has to say.

**Scene 1**

- *Section 1-4*: Robocchio is chased by a trash-collecting spaceship, so he is very scared and unsure about where to go. The eyes first move up and down (like arms asking for help) and then widen and close (indicating confusion). The body moves back and forth to indicate confusion, but also an attempt to use strength to escape the ship. The speed of these movements is high because of the agitated situation. The base of the robot moves around the stage also seeking in vain for shelter behind an object, again indicating fear and confusion. In its lines, the robot asks for help in vain. The robot is **scared**.

- *Section 5*: The robot is absorbed by the spaceship and it screams and wags its eyes back and forth. The robot moves backward quickly and ends up behind the scenes.

**Scene 2**

- *Section 1-2*: We are in the belly of the spaceship, the robot is scared and does not know where it is. He goes to the center of the scene and looks around at first moving his eyes and then the whole base.

- *Section 3*: Realizing it is in danger it starts to cry, then it lowers its eyes and bends its body down with medium speed. The robot is **sad**.

- *Section 4*: The robot hears a voice in the distance: it is the vacuum cleaner. He whirls towards her with **astonishment**.

- *Section 5-8*: The robot talks with the vacuum cleaner. It is **sad**.

- *Section 9*: Robocchio disagrees with what the vacuum cleaner says and shakes his eyes as if to say no.

- *Section 10*: Robocchio is still crying when he sees a light in the distance. It turns its eyes very **surprised**.

**Scene 3**

- *Section 1-2*: Robocchio returns to the center of the scene by moving his eyes left and right looking around. When he sees Gepp'Hett he quickly goes backward and widens his eyes showing a big **surprise**.

- *Section 3*: The robot is very **happy** to have found his father, he turns the base left and right, the torso swings, and the eyes move towards each other (like arms exulting).

- *Section 4-7*: Robocchio listens to Gepp'Hett's story, gets sad in bad moments by lowering his eyes, and reacts with amazement when Gepp'Hett tells him how he managed to survive. She approaches him slowly and moves her eyes up and down showing **curiosity**.

- *Section 8-9*: Robocchio with **courage** thinks of a solution to save himself. He raises his eyes and stiffens.

- *Section 10*: Gepp'Hett sticks to Robocchio (putting a hand on him) and together they leave the scene.

## 14.3 Results

The video has been uploaded to YouTube and can be found at the link:
*"https://www.youtube.com/watch?v=v7QM1OyLGcA&t=264s"*.

### 14.3.1 Comment on the performance of the robot

The video was recorded in the AIRLab conference room. The room was modified by covering the table with a cloth and adding pieces of cardboard. This is because the room was too symmetrical and the robot had difficulty locating itself, adding these asymmetries helped it locate itself perfectly. The video was shot dividing the three scenes, not in a single take. The rehearsal time was reduced, which led to some small errors. The robot was completely autonomous for the entire duration of the video following the script without the need for human intervention, except to decide the time in which to enter and exit the scene. At the point where the robot says: "E dopo?" there was an inaccuracy: the robot waited 2 seconds too much to say its line. This imperfection was caused by the fact that the previous line said by Gepp'Hett is very long and with many pauses, this led to having put a very long waiting time, which was adjusted by rehearsing once before recording. During the recording, Gepp'Hett said his line faster than expected and then led the robot to wait a few more seconds. We believe that the error is acceptable and can be solved by rehearsing the video for longer and adjusting the waiting times with more precision.

### 14.3.2 Survey

People were asked to watch the video and answer a few questions. The answers were 82, 52 males and 30 females with the majority of people being between 13 and 24 years. Half of the people were interested in theatre, and 70% are interested in robotics. The 65% of the people had no direct experience with a robot or have seen only movies, while the other 35% have interacted with a robot or have built one. The results are listed below.

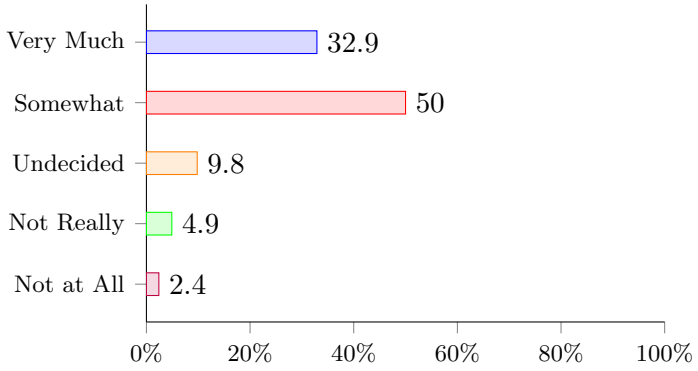



Figure 14.1: How much did you like the video?

These results show us that the public generally liked the video (Figure 14.1) and that the robot is credible in the representation of his character (Figure 14.2). Furthermore, most people believe that the robot can participate in a real show (Figure 14.3).

We also asked what emotions the robot showed during the video (Figure 14.4).

The results are very satisfying. With the robot, we wanted to represent 6 main emotions (fear, sadness, courage, surprise, happiness, curiosity). On average, 3.5 emotions were recognized per person and the most voted ones (fear, sadness, courage, surprise, happiness, curiosity) are the same ones that we wanted to represent. People also recognized some emotions that were not meant to be represented, but we think

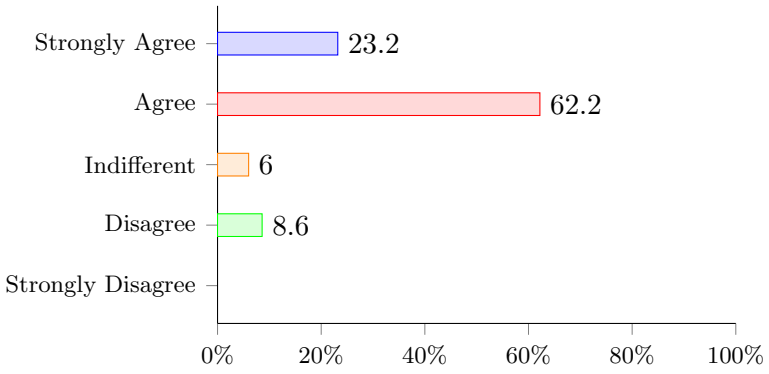



Figure 14.2: Do you think the robot is credible in the representation of his character?

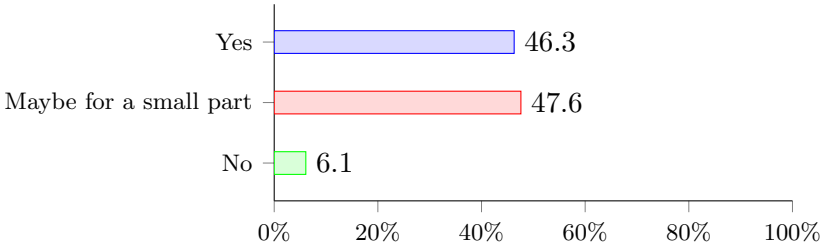



Figure 14.3: Do you think the robot who plays Robocchio could do it in a show at the theater?

that it was possible to detect (sympathy, shyness, anguish and anger). We also add some possibilities for emotions we think are completely wrong (shame, envy, embarrassment, disgust and boredom) and only a few bunches of people selected them. Furthermore, a certain distance can be noted between these 6 emotions and the others, further indicating that the emotions were clearly recognizable.

Finally, the results show us that our video makes both the world of robotics (Figure 14.6 and that of the theatre (Figure 14.5) more interesting, encouraging people to deepen these two worlds.

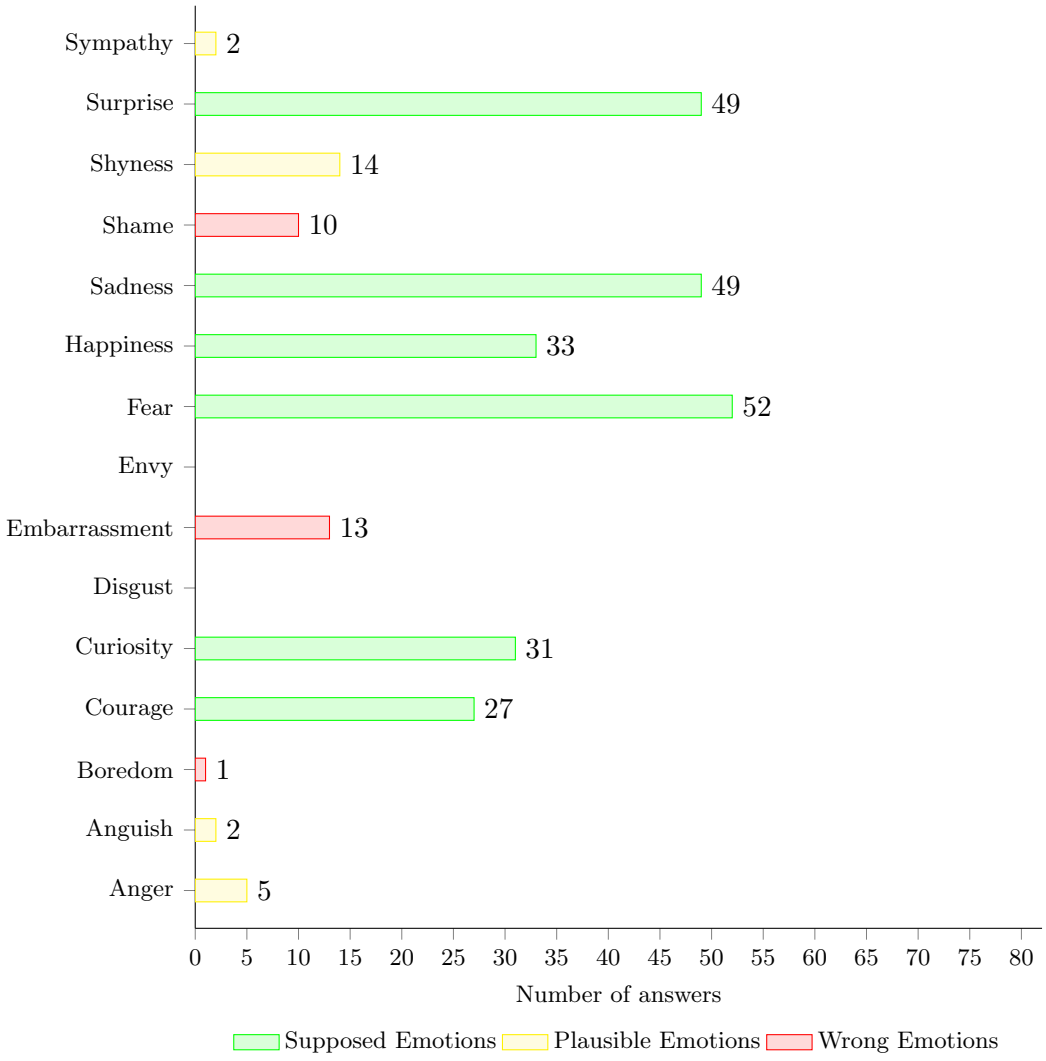



Figure 14.4: What emotions do you think the robot expressed during the video? (more than one answer possible)

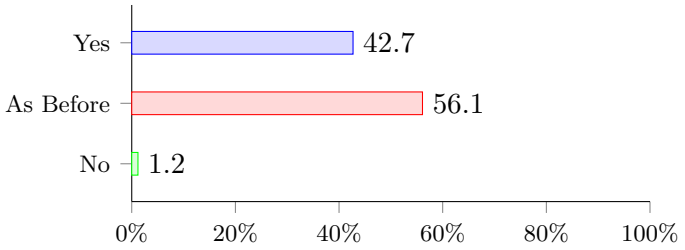



Figure 14.5: After seeing Robocchio, are you more interested in the world of robotics?

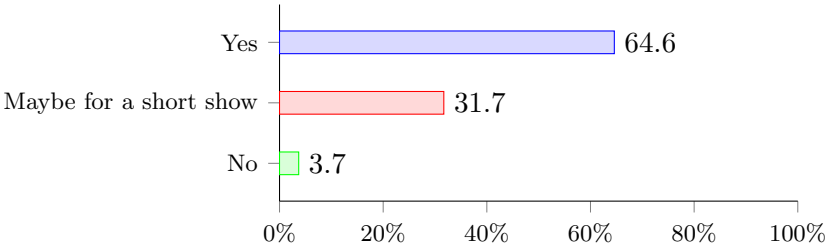



Figure 14.6: If you knew that there is a robot actor in a play, would you be more interested in that show?

# Chapter 15

# Conclusions and Future Works

## 15.1 Conclusions

This project ends showing positive results and many points for reflection and growth. Our robot can be used to produce an interesting stage. The robot amazes its audience from a technological point of view: it is incredible how in recent years we have gone from simple and sometimes repetitive animatronics to machines capable of innovating and changing the way of behaving in each show. Robocchio entertains his audience very well also from an expressive point of view. Thanks to its funny appearance and a skillful ability to make the most of its expressive abilities, the public of young and old is fascinated. Our project demonstrates how it is possible to create an interesting prop (or perhaps it would be better to call it a robot actor) while keeping the level of complexity (and therefore costs) very low. It will also be possible to use this type of robot for different tasks outside the theatre. It can be used as a platform to test emotional responses to certain human inputs and applied in areas where emotional communication with humans is also required. For example, it can be used in the medical field, helping doctors and nurses to take care of their patients by providing a dose of sympathy. It could be used for educational purposes, offering children to program it with the help of adults, bringing them closer to the world of robotics and theater. We would like to close this thesis with a reflection. Artificial intelligence is sometimes seen as an entity that wants to subdue the man and replace him in his duties (just think of works like I Robot and Blade Runner), but in reality, artificial intelligence is only a product of human intelligence, therefore it is not the computer that is intelligent, but the person who made it. For this reason, we must see robotics and artificial intelligence not as frightening enemies to be contrasted, but as an opportunity to contribute to the improvement of humanity. If it is true that technology will cause some people to lose their jobs, because they will be increasingly replaced by machines more efficient than them, it is also true that the technology to create these machines requires humans to be developed. As time passes, a balance will be created in which humans and machines will collaborate with each other using the skills in which each one excels. Furthermore, the possibilities of applying technology will not be limited to production, but also to art, and human thought will be conditioned by it. In fact, our robot can be seen as a form of art. As a sculptor expresses his thoughts and emotions through his statues, so an artist can express himself through the robots he builds.

## 15.2 Future Works

The robot wants to be considered a basic platform to which more complex functions can be added. Thanks to its architecture, in fact, it is possible to modify one of its functions making it more advanced or add a new one while maintaining the other basic functions. it is, therefore, an excellent starting point for developing further projects. Here are some ideas.

### Emotional Text to Speech

As mentioned in section 8.1 there is no software on the market able to act and express a significant amount of emotions. If an emotional text to speech is implemented we propose two possibilities to mount it on the robot:

- Implement an emotional text to speech able to generate audio files and replace them with the ones recorded.
- Implement an emotional dynamic text to speech able to generate emotional voices at run time and replace the code for reproducing the audio files, but keep the action server call.

### Emotional Move Base

The robot is able to move the base following the command of the operator or the script, but move_base reproduces paths without giving emotions to them. A possible implementation of this robot is to develop a new version of the planner to produce a path that expresses emotions, for example, it might change the velocities during the movement or change the shape of the path. This function could be added to the robot software by replacing the main controller's move base call with a new action server.

### Speaker Recognition

The robot follows the speech thanks to its microphone which allows it to understand if someone is speaking. An improvement would be to implement software, thanks to machine learning, that recognizes the voice of people in such a way as to recognize who is speaking. This would eliminate background noise influences such as applause, music and engine noise and would therefore allow the robot to make more movements even while others are talking without them disturbing the listening. It is also possible to add a microphone such as BlueCoin [30] that allows the robot not only to know who is speaking, but from which direction, so that it can turn exactly towards its interlocutor.

### Dynamic Emotions

Finally, the robot can be used as a test platform to test emotion-generating software from external inputs. For example, it is possible to install a camera on the head that observes the interlocutor, interprets his emotions and generates movements accordingly. In this case, the main controller would have to be replaced, but all the other action servers that control the robot's movements would be kept.

# Bibliography


[1] URL: https://www.softbankrobotics.com/emea/en/nao. [accessed 15-Marc-2021].

[2] URL: https://us.aibo.com/. [accessed 15-Marc-2021].

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

[4] URL: https://www.frieze.com/article/playing-real-eerie-prognoses-rimini-protokolls-new-robotic-theatrical-production-uncanny. [accessed 27-Marc-2021].

[5] URL: http://wiki.ros.org/.

[6] URL: http://wiki.ros.org/noetic.

[7] URL: http://wiki.ros.org/ROS/Concepts.

[8] URL: http://wiki.ros.org/actionlib.

[9] URL: http://www1.shuttle.eu/products/discontinued/barebones/dh310/.

[10] URL: https://store.arduino.cc/arduino-uno-rev3.

[11] URL: http://docs.ros.org/en/melodic/api/nav_msgs/html/msg/Odometry.html.

[12] URL: https://wiki.ros.org/navigation/Tutorials/RobotSetup/TF.

[13] URL: http://wiki.ros.org/gmapping.

[14] URL: http://wiki.ros.org/amcl.

[15] URL: http://wiki.ros.org/move_base.

[16] URL: https://wiki.ros.org/dwa_local_planner.

[17] URL: https://pypi.org/project/pyttsx3.

[18] URL: https://www.sonantic.io/.

[19] URL: https://www.readspeaker.com/solutions/text-to-speech-software.

[20] URL: https://cloud.google.com/text-to-speech/?hl=it.

[21] URL: http://wiki.ros.org/rosserial.

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

[23] URL: http://wiki.ros.org/urg_node.

[24] URL: http://docs.ros.org/en/melodic/api/std_msgs/html/msg/Int8MultiArray.html.

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

[26] URL: http://wiki.ros.org/rosserial_arduino/Tutorials/Arduino%5C%20IDE%5C%20Setup.

[27] URL: http://wiki.ros.org/joy.

[28] URL: https://pypi.org/project/PyAudio/.

[29] URL: http://wiki.ros.org/rviz.

[30] URL: https://www.st.com/en/evaluation-tools/steval-bcnkt01v1.html. [accessed 15-Marc-2021].

[31] 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.

[32] Jaclyn Barnes et al. "Robot Theater with Children for STEAM Education". In: *Proceedings of the Human Factors and Ergonomics Society Annual Meeting* 61 (Sept. 2017), pp. 875–879. DOI: 10.1177/1541931213601511.

[33] 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.

[34] Cynthia Breazeal et al. "Interactive robot theatre". In: vol. 46. Nov. 2003, 3648–3655 vol.3. ISBN: 0-7803-7860-1. DOI: 10.1109/IROS.2003.1249722.

[35] Albert van Breemen. "Animation engine for believable interactive user-interface robots". In: vol. 3. Jan. 2004, 2873–2878 vol.3. ISBN: 0-7803-8463-6. DOI: 10.1109/IROS.2004.1389845.

[36] Frank Broz et al. "Robot improv: Using drama to create believable agents". In: vol. 4. Feb. 2000, 4002–4008 vol.4. ISBN: 0-7803-5886-4. DOI: 10.1109/ROBOT.2000.845355.

[37] Carlo Lorenzini detto Collodi. *Le avventure di Pinocchio. Storia di un burattino.* 1883.

[38] Paul Ekman. "An Argument For Basic Emotions". In: *Cognition & Emotion* 6 (May 1992), pp. 169–200. DOI: 10.1080/02699939208411068.

[39] *Emotion.* URL: https://en.wikipedia.org/wiki/Emotion. Last modified March. 2021. [accessed 15-Marc-2021].

[40] Abrar Fallatah, Jeremy Urann, and Heather Knight. "The Robot Show Must Go On: Effective Responses to Robot Failures". In: Oct. 2019. DOI: 10.1109/IROS40897.2019.8967854.

[41] Naomi Fitter. *What's the Deal With Robot Comedy? How to teach a robot to be a stand-up comedian.* 2020. URL: https://spectrum.ieee.org/automaton/robotics/artificial-intelligence/whats-the-deal-with-robot-comedy.

[42] 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.

[43] Guy Hoffman. "OpenWoZ: A Runtime-Configurable Wizard-of-Oz Framework for Human-Robot Interaction". In: Mar. 2016.

[44] 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: 978-1-4244-2212-8. DOI: 10.1109/ROMAN.2008.4600691.

[45] 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.

[46] Ollie Johnston and Frank Thomas. "The Illusion of Life: Disney Animation". In: Oct. 1995, p. 576. ISBN: 0786860707.

[47] Franz Kafka. *La Metamorfosi*. 1915.

[48] 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.

[49] Rudolf Laban. *Modern Educational Dance*. Jan. 1963.

[50] David Lu. "Ontology of Robot Theatre". In: (Mar. 2021).

[51] David Lu and William Smart. "Human-robot interactions as theatre". In: Sept. 2011, pp. 473–478. DOI: 10.1109/ROMAN.2011.6005241.

[52] M Mori. "Bukimi no tani [the Uncanny Valley]". In: *Energy* 7 (Jan. 1970), pp. 33–35.

[53] Izabella Pluta. "Teatro e robótica: os androides de Hiroshi Ishiguro, em encenações de Oriza Hirata". In: 3 (May 2016), pp. 65–79. DOI: 10.36025/arj.v3i1.8405.

[54] R. Plutchik. "A General Psychoevolutionary Theory of Emotion". In: *Emotion: Theory, research, and experience* 1 (Jan. 1980).

[55] Tiago Ribeiro and Ana Paiva. "The Illusion of Robotic Life: Principles and practices of animation for robots". In: Mar. 2012. DOI: 10.1145/2157689.2157814.

[56] *Roboscopie, the robot takes the stage!* 2011. URL: https://www.openrobots.org/wiki/roboscopie. [accessed 15-Marc-2021].

[57] Krisztina Rosner. *The gaze of the robot: Oriza Hirata's robot theatre*. URL: https://thetheatretimes.com/the-gaze-of-the-robot/. [accessed 15-Marc-2021].

[58] James Russell. "A Circumplex Model of Affect". In: *Journal of Personality and Social Psychology* 39 (Dec. 1980), pp. 1161–1178. DOI: 10.1037/h0077714.

[59] Giuseppe Luigi Leandro Russo and Andrea Bonarini. "Model for social human-robot interactions and application to the theatrical context". 2016.

[60] Konstantin S. Stanislavskij. *Il lavoro dell'attore su se stesso*. 1938.

[61] 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.

[62] Louis Tassinary, John Cacioppo, and Gary Berntson. *Handbook of Psychophysiology*. Jan. 2017. ISBN: 978-1-107-05852-1.

[63] Silvan S Tomkins. "Affect theory". In: *Approaches to emotion* (1984), pp. 163–195.

[64] Michael Walters et al. "Companion robots for elderly people: Using theatre to investigate potential users' views". In: Aug. 2013, pp. 691–696. DOI: 10.1109/ROMAN.2013.6628393.

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

[66] 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.

[67] Kaiyu Zheng. “ROS Navigation Tuning Guide”. In: (June 2017).

# Appendix A

# Structure of the Workspace

This appendix contains descriptions of the ROS packages and their contents. The files related to the play specification have been omitted because they are not relevant.

Each ROS package contains:

- *package.xml*: the description of the package for the compiler.
- *CMakeList.txt*: the instructions for the compiler.

**main_controller**: controller of all the actions of the robot during the play

- *audio_files*: contains the audio files used for the play
- *cfg*: contains the config file for dynamic reconfigure
  - *main_controller.cfg*: configuration file for dynamic reconfigure
- *launch*: contains launch files for main_controller
  - *main_controller.launch* launch file for the main controller.
- *params*: contains the parameters files for the main controller
  - *main_controller_params.yaml*: parameter file for the main controller
- *scritps*: contains the scripts for the play
- *src*: source code
  - *main.py* pyhton code of the main controller

**speech_monitor**:

- *launch*: contains the launch file
  - *speech_monitor.launch*: the launch file
- *params*: contains the parameter file
  - *speech_monitor_params.yaml*: the parameter file
- *src*: contains the source code
  - *main.py*: the source code
  - *audio_analyzer.py* code for testing the parameters.

**audio_player**:

- *launch*: contains the launch file
  - *audio_player.launch*: the launch file
- *params*: contains the parameter file
  - *audio_player_params.yaml*: the parameter file
- *src*: contains the source code
  - *main.py*: the source code

**eyes_manager**:

- *src*: contains the source code
  - *main.py*: the source code

**body_manager**:

- *src*: contains the source code
  - *main.py*: the source code

**cmd_vel_manager**:

- *launch*: contains the launch file
  - *cmd_vel_manager.launch*: the launch file
- *src*: contains the source code
  - *main.py*: the source code

**triskarone_msgs**: contains the descriptions of custom actions messages.

- *action*:
  - *manual_move_base.action*: action description for the "cmd_vel_manager" action server.
  - *move_body.action*: action description for the "body_manager" action server.
  - *move_eyes.action*: action description for the "eyes_manager" action server.
  - *play_audio.action*: action description for the "audio_player" action server.
  - *speech_monitor.action*: action description for the "speech_monitor" action server.

**ira_laser_tools**:the package for the laser_data merger.

- *cfg*:contains the files for dynamic reconfigure
  - *laser_scan_multimerger.cfg*: the multimerger configuration file
  - *laser_scan_virtualizer.cfg*: the virtualizer configuration file
- *launch*: contains the launch files

  - *laser_scan_multimerger.launch*: the multimerger launch file
  - *laser_scan_virtualizer.launch*: the virtualizer launch file

- *src*: contains the source codes
  - *laser_scan_multimerger.py*: the multimerger source code
  - *laser_scan_virtualizer.py*: the virtualizer source code

**joystick**: the package for the joystick node.

- *include*: contains the header file.
  - *joystick_node.h* the header file
- *launch*: contains the launch file
  - *joystick.launch*: the launch file
- *params*: contains the parameter file
  - *joystick_params.yaml*: the parameter file
- *src*: contains the source code
  - *joystick_node.cpp*: the source code

**rosserial**: contains the ROS package: "rosserial", see ROS wiki for further description **odometry_publisher**: the package for the odometry node.

- *src*: contains the source code
  - *odometry_publisher.cpp*: the source code

**triskar_navigation**:

- *config*: contains the maps
- *launch*: contains the launch files for the navigation
  - *amcl.launch*: launch file for amcl.
  - *costmap_params.launch*: launch file for upload costmap params.
  - *gmapping.launch* launch file for gmapping
  - *map_server.launch* launch file for map_server
  - *move_base.launch* launch file for move_base
  - *slam.launch* launch file that starts all the nodes for slam
- *params*:
  - *amcl_params.yaml*: parameter file for amcl
  - *costmap_common_params.yaml*: parameter file for all the costmaps
  - *global_costmap_params.yaml*: parameter file global costmap
  - *local_costmap_params.yaml*: parameter file local costmap
  - *global_planner_params.yaml* parameter file for global planner
  - *navfn_params.yaml* parameter file for navfn

  - *gmapping_params.yaml* parameter file for gmapping
  - *base_local_planner_params.yaml* parameter file for base_local_planner
  - *dwa_local_planner_params.yaml* parameter file for dwa_local_planner
  - *move_base_general_params.yaml*: general parameters for move_base
  - *clear_costmap_params.yaml* recovery behaviour parameters
  - *rotate_recovery_params.yaml* recovery behaviour parameters

**urg_node**: node for controlling the lasers, see ROS_wiki for further information
**triskar**: contains the launch files for the robot

- *launch*: contains the launch files
  - *arduino.launch* launches arduino nodes
  - *audio.launch* launches audio nodes
  - *hokuyo_nodes.launch* launches the lasers nodes
  - *triskarone.launch* launches all the nodes for controlling the robot (a part from main_controller and slam)
  - *triskarone_slam.launch* launches all the nodes for controlling the robot (a part from main_controller)

# Appendix B

# User Manual

## B.1 Set up the code

### Download

You can download the robot code at the link:
*https://github.com/LorenzoBonetti/RoboTheatreWS* To make the code work you have to install the following packages (from terminal):

- *ROS Noetic*: sudo apt install ros-noetic-desktop-full
- *Move_Base*: sudo apt install ros-noetic-move-base
- *Gmapping*: sudo apt install ros-noetic-slam-gmapping
- *Amcl*: sudo apt install ros-noetic-amcl
- *pyaudio*: pip3 install pyaudio
- *joy_driver*: sudo apt install ros-noetic-joy
- *laser_driver*: sudo apt install ros-noetic-urgnode

### Udev Rules

Most of the external sensors and devices are connected trough USB ports. We used a couple of rules to rename the devices and give them enough privileges to run correctly. You can found these rules and how to apply them here https:// github.com/AIRLab-POLIMI/Robotower2.0/tree/master/core/triskarone/ triskar/udev

### Set up

1. Create a new folder and then copy the *"src"* folder inside.
2. Compile the code with the command *catkin_make*
3. Type *gedit /.bashrc*, then add a new line *source path_to_your_workspace/devel /setup.bash*
4. Check the IP of the robot (ifconfig -a) and change the values of the variables: *ROS_MASTER_URI* and *ROS_IP*

## B.2 How to start the robot

On your PC connect with the robot via SSH (id: airlab pw: aerolabio), then start up the robot:

1. *roslaunch triskar triskarone_slam.launch*

2. *roslaunch triskar_navigation move_base.launch*

On a terminal not connected via SSH open rviz. Localize the robot on the map using the command on rviz and move the robot around to be sure the robot is correctly localized, then start the main controller to begin the script: *roslaunch main_controller main_controller.launch*

## B.3 Possible failures and how to fix them

This section describes some faults that may occur and the solution to solve them.

#### The computer does not turn on

Check that the batteries are charged and connected to the power supply, check that the switch is on. You may have touched inside the power cord in the bottom center, touch it gently to put it back in place.

#### The computer freezes after entering the password when is connected to the monitor

It is possible that the sound card did not start up correctly, unplug the HDMI cable from the computer and try to reconnect it or turn it off and on the pc. If it doesn't work, repeat the operation

#### The robot does not play audio

It is possible that the sound card has not started correctly, connect the PC to the monitor and check that the audio settings are correct, then start a random audio

#### The robot does not move the base

If the robot does not move check that the motor switch is on. Check that power is supplied to the motors (the LED on the emergency button is on), if it is off, check that the emergency button is not pressed, otherwise check that any cables in the power supply box have not been disconnected. If it is a driver problem look at the next point

#### Eyes or body do not move

Check that all servo motor cables are properly connected to power and Arduino. It may have happened that one of the motors has broken (due to an overload), check the operation of each motor and replace the broken one. In the case of the eyes, check that no cables have come out of the guides or have broken.

### Nova Core drivers are not properly connected or synchronized

It may happen that an error message appears on the terminal indicating a connection error with the drivers. You may notice that the drivers are not working correctly when all 4 LEDs are not blinking at the same time. To solve it, turn off all and how many ROS nodes, disconnect and reattach the USB cable of the drivers, press on all 4 white connectors. If it doesn't work, repeat the operation

### Lasers don't work

It may happen that you get error messages related to a failed connection of the lasers. Restart the program, if it doesn't work, unplug and reattach the USB cables of the lasers.

### Move Base doesn't work

If the robot cannot reach the destination there is a problem with the planner. First of all, check that some part of the robot has not detached obstructing the laser field of view (on rviz you can see a fixed obstacle near the robot).

If the problem is the setting of the parameters, adjust them following the guide in the chapter 7

### The robot cannot connect to the master node

Check if the router is turned on. Check that the ROS Master IP is correct:

1. Open the terminal
2. Type *ifconfig-a* to know the robot's IP address.
3. Type *gedit /.bashrc* and check if *ROS_MASTER_URI* and *ROS_IP* have the correct value.

# Appendix C

# Complete Scripts

Here is reported the script of the play and a piece of the robot's script.

## C.1 Script

**Narratore**: Robocchio, un robot costruito dall'ingegner Gepp'Hett, vorrebbe essere un bambino vero: dopo essere scappato dal laboratorio in cui è nato, ha affrontato mille peripezie. Derubato dei suoi BitCoin dall'Astrogatto e la Firevolpe, incontrato il mangiacorrente e visitato il Saturn Park, ora è in cerca di suo padre in compagnia di Alessia, l'assistente turchina. All'improvviso dalle stelle appare un'astronave raccogli rifiuti, l'incubo di ogni macchina senza più aggiornamenti...

#### Scena1

**Robocchio1**:Aiuto! Aiuto! un mostro mi insegue!
**Alessia**: Affrettati Robocchio per carità
**Robocchio2**: Vai via, lasciami stare
**Alessia**: Sta attivando il raggio traente!
**Robocchio3** Panf Panf non ce la faccio più
**Alessia**: Bada Robocchio! il gigante ti raccoglie! Eccolo! Eccolo!

#### Scena2

*Cambio scena, ponte dell'astronave, Robocchio si guarda intorno*
**Robocchio4**: Aiuto! Aiuto! Oh povero me! Non c'è nessuno che venga a salvarmi?
**Aspirapolvere**: Chi vuoi che ti salvi disgraziato?
*Pinocchio alza gli occhi di scatto, qui abbiamo sorpesa*
**Robocchio5**: (sospiro di stupore) Chi è che parla così?
**Aspirapolvere**: Sono Io! Una povera aspirapolvere, inghiottita dall'astronave insieme a te. E tu che elettrodomestico sei?
**Robocchio6**: Io non ho a che vedere nulla con gli elettrodomestici. Io sono un robot.
**Aspirapolvere**: E allora, se non sei un elettrodomestico, perchè ti sei fatto assorbire da questa astronave?
**Robocchio7**: Non sono io che mi sono fatto assorbire, è lei che mi ha assorbito! E ora che dobbiamo fare qui al buio?
**Aspirapolvere**: Rassegnarci e aspettare che l'astronave ci abbia smaltiti tutti e due!
*Pinocchio riabbassa gli occhi e busto e si rimette a piangere*
**Robocchio8**: Ma io non voglio essere smaltito!

**Aspirapolvere**: Neppure io vorrei esserlo, ma io sono abbastanza filosofo e mi consolo pensando che quando si nasce aspirapolveri, c'è più dignità a morir riciclato che sotterrato
**Robocchio9**: Sciocchezze
**Aspirapolvere**: La mia è un opinione, e le opinioni vanno rispettate.
**Robocchio10**: Insomma...io voglio andarmene di qui...io voglio fuggire...
**Aspirapolvere**: Fuggi se ti riesce!
*Si accende una luce in lontananza*
**Robocchio11**: (espressione di sorpresa) Che cosa sarà mai quel lumicino lontanto lontano?
**Aspirapolvere**: Sarà qualche nostro compagno di sventura, che aspetterà come noi il momento di essere smaltito!...
**Robocchio12**:Voglio andare a trovarlo. Non potrebbe darsi il caso che sia qualche vecchio computer capace di insegnari la strada per fuggire?
**Aspirapolvere**: Te lo auguro con tutto il cuore caro robottino.
**Robocchio13**: Addio Aspirapolvere
**Aspirapolvere**: Addio robottino, e buona fortuna!

**Scena 3**

*Cambio di scena, Robocchio si guarda in giro, entra Gepp'Hett*
**Robocchio14**: Oh babbo mio! Finalmente vi ho ritrovato! Ora non vi lascio più, mai più
**Gepp'Hett**: Dunque gli occhi mi dicono il vero? Dunque sei proprio tu il mio caro Robocchio?
**Robocchio15**: Sì, sì, sono io, proprio io! E tu mi hai già perdonato non è vero? Oh babbo mio come sei buono! e pensare che io...invece...Oh! ma se sapessi quante disgrazie mi sono piovute sul capo e quante cose che mi sono andate per traverso.
E quant'è che sei chiuso qui dentro?
**Gepp'Hett**: Da due anni, due anni Robocchio mio che sono parsi secoli!
**Robocchio16**: E come avete fatto a campare? E dove avete trovato la torcia? E le batterie per accenderla? Chi te li ha dati?
**Gepp'Hett**: Ora ti racconterò tutto. Devi sapere che quell'asteroide che distrusse la mia nave colpì anche una astrocorriera di Barattolini, i corrieri si salvarono, ma gli ordini non furono mai consegnati e i pacchi vennero inghiottiti da questa astronave
*Pinocchio è stupito, occhi da basso a medio come per strabuzzarli, busto in avanti*
**Robocchio17**: Come? li inghiottì tutto in un boccone?
**Gepp'Hett**: Tutto in un boccone. Per mia gran fortuna, quei pacchi erano carichi di carne in latta, biscotti, di bottiglie di vino, di scatole di batterie e torce. Con tutta questa grazia di Dio ho potuto campare due anni: ma oggi sono agli ultimi sgoccioli: oggi nella dispensa non c'è più nulla, e questa batteria, che vedi accesa, è l'ultima che mi è rimasta.
*Pinocchio un po' intimorito*
**Robocchio18**: E dopo?
**Gepp'Hett**: E dopo, caro mio, rimarremo tutti e due al buio.
**Robocchio19**: Allora, babbo mio, non c'è tempo da perdere. Bisogna pensare subito a fuggire...
**Gepp'Hett**: A fuggire? E come?
**Robocchio20**: Scappando dalla bocca del portello di ingresso e rubando una navetta di soccorso. Attaccati a me!
*Pinocchio si muove con Geppetto che lo segue ed escono di scena*

## C.2 Robot Script

Here is reported an example of the script of the robot, only the scene 3 is reported for simplicity

**Scene 3**


```
{
"section1": {
        "trigger": "after_command",
        "trigger_data": 1,
        "actions": {
                "move_eyes":[1,0,6,0,1,3,1,2,3,1,1,3,1,0,3],
                "move_body":[2,0,3],
                "move_base":[0.61,-0.24,0.58,0.82]
        }
},
"section2": {
        "trigger": "after_precedent",
        "trigger_data": 0.5,
        "actions": {
                "manual_move":[1.5,0.0,0.0,-1.0,0.0,0.0,0.1,1.0,0.0,0.0,0.5,0.0,0.0,2.0],
                "speak":"urlo4.wav",
                "move_eyes":[0,6,2,3,0,2],
                "move_body":[0,4,2,3,0,2]
        }
},
"section3": {
        "trigger": "after_precedent",
        "trigger_data": 0.5,
        "actions": {
                "speak":"robocchio14.wav",
                "manual_move":[0.0,0.0,1.0,0.0,0.0,-1.0,0.5,0.0,0.0,2.0,0.0,0.0,1.0,0.5,
                0.0,0.0,1.8,0.0,0.0,-1.0,0.5],
                "move_eyes":[0,4,2,1,0,2,1,4,2,1,0,2],
                "move_body":[0,2,2,1,3,2,1,0,2]
        }
},
"section4": {
        "trigger": "after_speech",
        "trigger_data": 1.5,
        "actions": {
                "speak":"robocchio15.wav",
                "move_eyes":[0,3,2,0,0,2,0,3,2,0,0,2,2,3,5,1,0,3,1,3,4],
                "move_body":[4,2,2,1,0,2,3,1,2]
        }
},

    "section5": {
        "trigger": "after_precedent",
        "trigger_data": 0.5,
```

```
        "actions": {
                "do_nothing":1
        }
},
"section6": {
        "trigger": "after_precedent",
        "trigger_data": 1.5,
        "actions": {
                "speak":"robocchio16.wav",
                "move_eyes":[0,0,2],
                "move_body":[0,0,2]
        }
},
"section7": {
        "trigger": "after_speech",
        "trigger_data": 3.5,
        "actions": {
                "speak":"robocchio17.wav",
                "move_eyes":[0,3,1,0,0,1,0,3,1,0,0,2],
                "move_body":[1,1,4,3,0,3]
        }
},
"section8": {
        "trigger": "after_speech",
        "trigger_data": 9.5,
        "actions": {
                "speak":"robocchio18.wav",
                "move_eyes":[0,4,1,2,0,1],
                "move_body":[0,4,4,2,0,3],
                "manual_move":[0.5,0.0,0.0,-0.25,0.0,0.0,0.5]
        }
},
"section9": {
        "trigger": "after_speech",
        "trigger_data": 22.5,
        "actions": {
                "speak":"robocchio19.wav",
                "manual_move":[0.5,0.0,0.0,0.25,0.0,0.0,0.5],
                "move_body":[0,1,3]
        }
},
"section10": {
        "trigger": "after_speech",
        "trigger_data": 1.5,
        "actions": {
                "speak":"robocchio20.wav",
                "move_eyes":[0,1,1,0,2,1,0,0,2]
        }
},
"section11": {
```

```
        "trigger": "after_speech",
        "trigger_data": 1.5,
        "actions": {
                "speak":"robocchio21.wav",
                "move_body":[0,2,2,0,3,2,0,0,2]
        }
},
"section12": {
        "trigger": "after_command",
        "trigger_data": 0.5,
        "actions": {
                "move_eyes":[2,0,6],
                "move_body":[2,0,3],
                "move_base":[0.76,2.22,-0.67,0.75]
        }
},
"section13": {          "trigger":"after_precedent",
        "trigger_data": -1,
        "actions": {
                "end": -1
        }
}
}
```