# HRI-RA Project: Pepper house assistant

Fabrizio Casadei - Matricola: 1952529


SAPIENZA
UNIVERSITÀ DI ROMA

September 20, 2023

# Contents

# Chapter 1

# Introduction

The idea of a robot working as a home helper in today's fast-paced society is not just science fiction, but a revolutionary reality. Modern robotics, artificial intelligence, and smart home technology have combined to create a new era of domestic life where robots serve as dependable housemates, caretakers, and problem solvers. This project wants to provide a case of study for the development of such intelligent systems, in order to improve the humans' quality of life.

## 1.1 HRI

The mutual dependency of people and machines is now more apparent than ever in a time of impressive technological and artificial intelligence progress. The study of *human-robot interaction* or *HRI* is evidence of how our relationships with intelligent beings are developing. We examine the complex interplay between humans and robots through the study of HRI, where machines serve as more than simply tools, they also serve as partners in collaboration and decision-making.

For example, intelligent robots with a good interaction with the humans degree can be used for:

- Can carry out repetitive, labor-intensive operations with accuracy and consistency, increasing productivity and efficiency. Humans may delegate these duties by engaging with robots
- Increasing Safety: Robots can carry out risky work in unsafe settings when people's safety is at risk. HRI lowers the risk of accidents and fatalities by allowing humans to work alongside robots in these settings
- Robots and intelligent agents may be programmed to recognize and react to the unique requirements and preferences of each individual human being, enabling personalized assistance. Whether in customer service, healthcare, or daily living, personalization of services may significantly improve quality of life.

This type of interaction includes several technologies like *Robotics*, *Artificial Intelligence*, *Computer vision*, *Machine Learning*, *NLP* (Natural Language Processing), and so on.
The quality of the relation between robot-human is really dependent on the levels of social skills that a system can provide [1], but depends on many other aspects like Robot morphology (*Uncanny Valley* [2]), adaptation level (ML), Audio & video processing, degree of autonomy, etc.
One of the key objectives of this part of the project is to reach a good level of interaction using **Pepper** robot (figure 1.1). In this scenario, Pepper is a house assistant that interacts and exchanges

information about the house and tasks that can be accomplished. Everything is developed in a simulated environment, both the interaction Human-Robot and the Robot-House one.
The interaction is multimodal and contextual to this scenario, combining different functionalities of the Robot (like simulated ASR) to offer a suitable demonstration.

## 1.2 RA

The project focuses not only on the interaction with the Robot but also on its ability to reason, using tools and Artificial Intelligence related techniques, to accomplish non-trivial tasks. The idea of reasoning agents systems endowed with the capacity to comprehend, interpret, and react to human actions, intents, and emotions, lies at the core of this developing dynamic.
By bringing an exceptional level of flexibility and context awareness to HRI, these Reasoning Agents driven by artificial intelligence yield interactions with robots more natural, effective, and influential.
In this scenario, we have used and designed powerful tools to perform deterministic planning, obtaining a solution of the task at a **high level**, then a **low level** algorithm based on *Artificial Potential Field* is used to perform motion planning. The motion is performed without defining any timing law, but rather the important aspect is to reach every time a target position, avoiding obstacles.
Thus, the goal for this part is to define an approach to realize the goal with an automated behavior synthesis. This is performed using action theories in deterministic settings.

## 1.3 Development

This project is developed using mainly coding Python files. To perform planning, *PDDL* files have been exploited and automatically adapted by the system (more on this later).
To interact with robot OS, NAOqi APIs have been used, through the SDK provided by *Softbank Robotics*.
In order to have a multimodal interaction, we interface with the tablet, started using MODIM (Multimodal Interaction Manager).
The robot simulator used is *Choregraphe* which allows us to analyze and represent the behavior of the robot in an almost realistic fashion based on the instructions provided.
Moreover, a huge work has been conducted, building from scratch a planar (2D) house simulator, used to show the resolution of tasks assigned, and useful to have a representation of the binding between the HRI and RA work.

## 1.4 Pepper

Pepper is a robot built to interact with people in a variety of social and service-related settings. It was created by Japanese robotics company SoftBank Robotics.
Pepper is a popular option for applications involving human interaction because of its welcoming design and expressive characteristics. Pepper is designed with the ability to recognize emotions through the detection and analysis of facial expressions and voice tones.
This Robot is equipped with a lot of sensors (figure 1.2): stereo cameras, four directional microphones, touch sensors, inertial sensors, and Collision-avoidance sensors (i.e. Ultrasound transmitters and

receivers).
Moreover, it includes within its body a tablet for visual presentations and tactile feedback, which can be used as another manner of interaction.

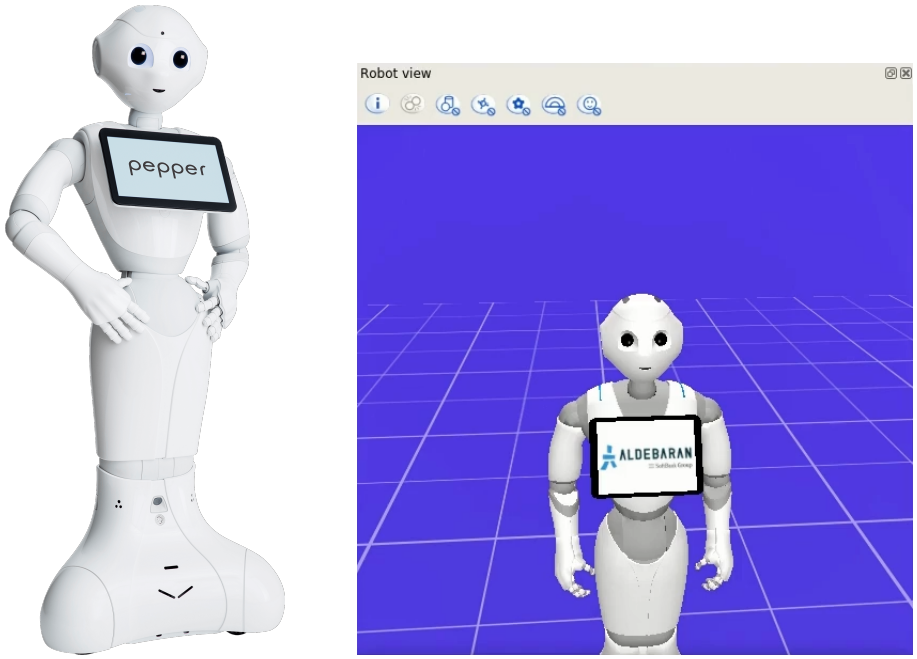


Figure 1.1: Pepper robot on the left and its 3D model by simulation on the right

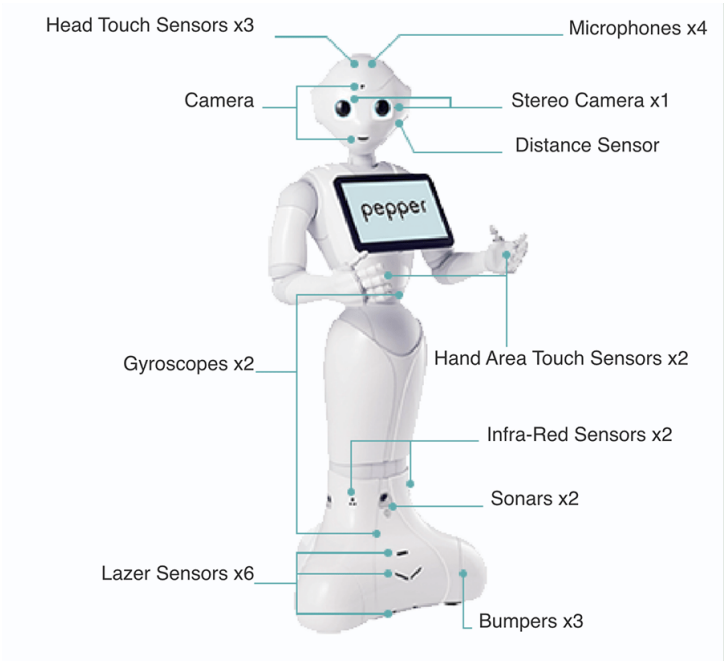


Figure 1.2: Pepper's sensors

# Chapter 2

# Related work

In this chapter, are presented and cited previous works and scientific publications related to this project, emphasizing separately the part of Human-robot interaction, and the one of Reasoning agents.

## 2.1 HRI

HRI is crucial in different contexts, for example, remaining to the one related to this project, home assistance. An important study has been carried out from [3], the research discussed in this paper looked at how people felt about the prospect of a future robot companion for the house. 28 adults' responses to questionnaires and experiments involving human-robot interaction were used to gather data using a human-centered approach. Results showed that a significant number of individuals supported robot companions and perceived their possible use as assistants, or servants. Few people want a robot as a companion. Child and animal care jobs were chosen above tasks at home. While humanlike behavior and appearance were less important, humanlike communication was ideal for a robot friend. Results are reviewed in regard to potential future study topics for the creation of robot companions.

Research carried out by Graf et al. [4] has as objective the creation of a mobile robot that can assist individuals in their homes. The most recent prototype, *Care-O-bot 3*, is a vision of a future home product and is fitted with the most advanced industrial hardware components. It includes all modern multimedia and interaction equipment as well as the most sophisticated sensors and controls. Care-O-bot 3 has been demonstrated to the public several times, serving beverages to attendees of events and trade shows. The robot will be used at an elderly care home to assist the staff in doing their everyday duties, according to recent developments.

Moving towards another field that makes massive use of HRI, we present a work connected to healthcare.
The area of robotics has grown significantly in recent years. In addition to other industries, robots are beginning to penetrate the healthcare industry. However, as the population of older people continues to increase, the present medical staff and healthcare professionals are forced to deal with a rising number of elderly patients, particularly those who have degenerative diseases like senile Alzheimer's, and other similar conditions. As a result, we may anticipate significant advancements in the field of medical robotics, particularly for that sub-category of robots known as *Socially Assistive Robots*

(SARs) that help to facilitate social interaction and companion users. The goal of SARs is to assist human users through social interaction by combining assistive robots with socially interactive robots. There have been attempts to create SARs for communication with dementia patients. The so-called social robots have evolved on several forms. Some of them imitate very simple animals, such NeCoRo, AIBO, and Paro [5].

Moving to the actual development, the work by Peltason & Britta [6] presents a way to rigorously define the link between dialog structure and task structure. They provide interleaving interaction patterns together with a generic protocol for task communication. Scalability is introduced and advanced dialog is handled.
The framework aims to enable the creation of useful spoken language applications for the actual world. These are frequently created, therefore the framework should allow strong application extension.
The proposed approach centers on the idea of interaction patterns, which serve as customizable (and hence reusable) interaction building blocks, offering fine-grained task state protocol that connects domain level and dialog, and a flexible dialog modeling is made possible via interleaving patterns (patterns that can be interrupted by other patterns and possibly resumed later).

## 2.2 RA

The application of deterministic planners, powered by symbolic representations like PDDL in this case, has long been an essential element in the fields of automated planning and artificial intelligence. Following the rules of traditional planning, these planners are excellent at locating solutions within well-defined domains by investigating numerous action sequences.
However, the limits of traditional deterministic planners become clear as planning tasks are broadened to include more dynamic and time-sensitive scenarios.
The concept of *Linear Temporal Logic* (LTL) is a powerful tool for expressing temporal properties and constraints in planning problems.
The work presented by Camacho et al. [7] Focuses on *LTL synthesis*, which involves generating a strategy that satisfies Linear Temporal Logic (LTL) specifications over infinite traces. Specifically, investigates LTLf synthesis, a variant tailored to scenarios where specifications are interpreted over finite traces.
Existing methods for LTLf synthesis typically transform LTLf into deterministic finite-state automata (DFA) and frame the synthesis problem as a DFA game. However, this DFA transformation incurs a worst-case double-exponential computational complexity, posing a significant computational challenge.
In contrast, the synthesis issue is changed into a non-deterministic planning problem that uses non-deterministic automata, solving this problem.

An alternative technique to implement an intelligent system using numeric PDDL is presented in [8]. Dealing with deadlines and action durations is a need in many real-world situations.
Making plans that require consideration of these limits, as well as propositional and resource circumstances. These characteristics are combined in this study by dealing with the associated temporal limited planning problem utilizing the numeric planning framework, which is based on an action-centered philosophy similar to the PDDL. The technique was developed because a multi-modal travel planning challenge necessitated the integration of scheduling and planning. This work introduces *JoPA*, a web service architecture. The paper shows how the majority of the temporal conditions can be translated by exploiting numeric fluents and constraints.

# Chapter 3

# Solution

In the present chapter, we outline the proposal made by this project to carry out the work to be realized in the domestic scenario.

## 3.1 HRI

Our work was carried out in the Linux environment, where *Docker* makes feasible to build one or more containers that can be used to run independently without affecting the system in any way. It is made possible by performing virtualization at the operating system level.
The initialization of the project requires several commands to be executed by terminal: run docker image and its related terminal multiplexer [1], launch NAOqi server, start choregraphe simulator, and launch the local *MODIM*(Multi-modal Interaction Manager) [2] server.
NAOqi is required, and its APis as well, since Pepper uses the NAOqi operating system.
Choregraphe is then configured to work with Pepper robot at a specified IP and Port (Localhost IP and 9559 Port), in order to correctly exchange data with NAOqi.
The MODIM server allows in this application the utilization of the tablet and thanks to the code developed, it interfaces through calls with the NAOqi module.
In the below figure 3.1, we have a schematic representation of this architecture.

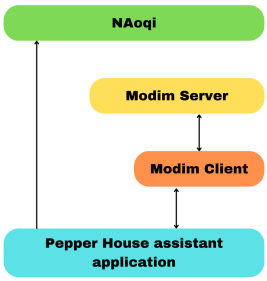


Figure 3.1: HRI software, general overall structure

[1] https://bitbucket.org/iocchi/hri_software/src/master/
[2] https://bitbucket.org/mtlazaro/modim/src/master/

Different types of tools or services provided by the SDK from Softbank are been used. In details:

- **ALMemory**, provides memory information related to the hardware configuration of the robot. More precisely, it offers details on the sensors' and actuators' most recent states.
  Moreover, this module serves as a central location for the dissemination of event alerts and may be used to store and retrieve named variables. It's useful for example to define touch interaction, activate actions driven by sonar, and define callbacks for the Human-Robot dialog.
- **ALMotion**, offers methods that facilitate making the robot move, in this project is used mainly to animate the robot, the motion is simulated directly on the House simulator.
- **ALRobotPosture**, allows the robot to realize different predefined postures, used for wakeup and sleep Pepper.
- **ALTextToSpeech**, grants the robot to speak. It sends commands to a text-to-speech engine, and authorizes also voice customization.
- **ALDialog**, used to endow your robot with conversational skills by using a list of "rules" written and categorized in an appropriate way. This set of rules is defined in .top file, defining the "Topic".
- **ALAudioPlayer**, provides playback services for multiple audio file formats and the associated common functionalities (play, pause, stop, loop, etc.)

This project also includes MODIM, a service that operates as a result of the creation of several readily invoked interaction functions. This service uses action files, composed with a certain fixed structure, useful for configuring the tablet experience. These files have different specifications:

- **IMAGE**, to show a representative image in the current view.
- **TEXT**, to write on the tablet's touch screen.
- **TTS**, allows the robot to say a given text.
- **ASR**, let the use of the Automatic Speech Recognition module of the robot.
- **BUTTON**, provide buttons that can be used to interact directly with the robot or to load another view/action file.

The action files include the possibility of configuring in a multilingual setting, providing text for the default language and the other specified.
The MODIM server is responsible of the connection between the HTML pages (what it's seen on the tablet) and the main application.

## 3.2 House simulator

To model and simulate robot motion in a 2D environment, we developed from scratch a custom 2D house simulator. The simulator serves as a crucial component of our experimental framework, allowing us to visualize and analyze the robot's movements in a controlled environment.
The simulator was constructed using the *Pygame* library, a widely used and versatile Python framework for developing 2D games and graphical applications.
This project component is central for both the HRI and RA parts since gives visive feedback on what's happening during the interaction of the robot with the House.
Considering the impossibility of handling the interaction of Pepper in a real house, we believe that this software provides a good compromise.

## 3.3 RA

As previously stated, The Pepper Agent, should be able to solve house tasks, like:

- **Navigation**, Pepper is able to autonomously reach any room starting from any position in the House. This can be seen as a classical Regulation problem, it's not important how it arrives at the goal position.
- **Interact with the environment**, Pepper should be able to open and close doors, and house windows as well. This action, for natural reasons, is extremely simplified, since it would necessitate of advanced image processing, computer vision, and visual-servoing techniques that in a planer 2D simulator it's not possible to develop.
- **Move object**, Pepper, acquired the necessary knowledge of the problem, should be capable of grasp, move and place an object on the specified house's furniture. Also in this case are valid the same simplification exposed in the previous point.
- **Avoid obstacles**, The robot should move avoiding obstacles, without having an internal map (i.e. From SLAM) of the house, instead using its proximity sensors and structured knowledge of the house. The facilitation in this case is present in the full localization of the robot and the targets.

This work is realized by dividing the problem into two parts: the first is about deterministic planning, so finding a plan at a high level, then the second is the interpretation and execution of the plan. The interpreter can be seen as a low-level planner, that takes the plan action and decomposes them into directives for the robot.
Above all, the plan interpreter performs motion planning, providing at each time the direction of motion to reach the goal and avoid collision with the environment.
The code of this part of the project is developed in Python and doesn't run on the docker container.
Let's examine these features in more depth.

### 3.3.1 Determinsitic Planning

Deterministic planning comprises a systematic examination of state space to find the best paths to reach a desired state. The state space, action space, beginning state, and intended target state

are crucial parts of this process. To efficiently explore these components, a variety of algorithmic techniques are used, including forward planning, backward planning, and heuristic search techniques like A*.
In this project has been used as a planner the *FF* or *Fast-Forward planning system* Developed by Jörg Hoffmann and Bernhard Nebe [9]. This is a popular and influential domain-independent heuristic-based planner used in the field of artificial intelligence and automated planning, which has demonstrated its effectiveness in various international planning competitions and benchmarks. This planner is heuristic-based and domain-independent, it relies on forward search in the state space, guided by a heuristic that estimates goal distances by ignoring delete lists.
More precisely, the planning system version used in this work is the*Metric-FF* [10], an extension of FF with more advanced capabilities.

The planner needs a description of the problem and its domain, which is incorporated into PDDL files.
The *Planning Domain Definition Language* (*PDDL*) is a crucial tool in deterministic planning, that provides a convenient way to represent problems of this type. Each planning task is defined by a problem file and a domain file.
The domain file provides the actions, predicates, and structure of the planning problem. While the problem file describes the details of the beginning state and goal conditions. This modeling process translates the problem into a form that deterministic planners can effectively explore. To effectively use these powerful tools in a dynamic way, has been defined a domain file that describes actions, types, and predicates. This file remains fixed during the whole execution. While the problem file is first defined with a template, that includes objects, initial states, and an empty goal. Then, a parser developed from scratch, generates a domain file based on this template, updating the initial states of the problem and the goal, based on past executions and current necessities.

### 3.3.2 Motion Planning

Given the high-level plan, we want to be able to move from one room to another without colliding with house obstacles, like furniture and walls.
Therefore, is required that the robot in autonomy is able to plan a collision- free motion from an initial to a final posture on the basis of geometric information about the workspace. The geometric information is provided by proximity sensors that allow the computation of the *clearance*, defined as the minimum distance between the robot in a certain configuration q and the boundary of free space in the room. Mathematically:

$$\lambda(q) = min_{s \in FreeSpace} \|q - s\|$$

In this case, approximating the Pepper primary workspace to a cylinder that contains the robot, we can use as configuration variable, the planar cartesian position, so $q = p = (x, y) \in \mathbb{R}^2$.
The algorithm should be online, information is gradually discovered by the robot during the navigation in the workspace.
There are many methods to perform motion planning, like *roadmap methods*, *probabilistic methods*, etc. In this case has been utilized the *Artificial Potential Field method* or (APF).

This type of method follows the stimulus-response paradigm or reactive navigation, which means that the system simply reacts to external stimuli.
The idea is simple: build a potential field in space so that the point that represents the robot is attracted by the goal and repelled by the obstacle regions.
The total potential $U$ is the sum of an attractive and a repulsive potential, whose negative gradient $-\nabla_q U$ indicates the most promising local direction of motion, the direction (using negated gradient) for the decreasing of the argument function $U(q)$. Defining the position error as the difference between the actual position and the goal: $e(q) = q_g - q = p_g - p$, the attractive potential can be defined in different ways, *tparaboloidal*, *conial*, or through a mixed solution that combines both (figure 3.2).

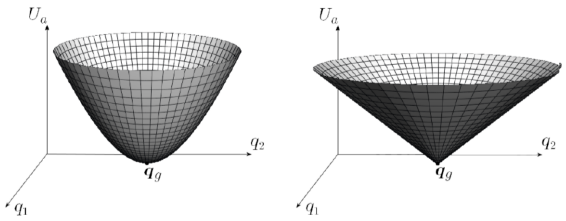


Figure 3.2: APF profiles shape

All these 3 profiles are been implemented. We now treat the mathematical aspects only of the conical one, defining the potential and the associated force.

$$U_{a,conical}(q) = k_a \|e(q)\|, \qquad with: k_a > 0$$

$$f_{a,conical}(q) = -\nabla_q U_{a,conical}(q) = k_a \frac{e(q)}{\|e(q)\|}$$

Instead for the repulsive potential, we define a potential and a force for each obstacle.

$$U_{r,i}(q) = \begin{cases} \frac{k_{r,i}}{\gamma}(\frac{1}{\eta_i(q)} - \frac{1}{\eta_{0,i}})^\gamma & if: \eta_i(q) \leq \eta_{0,i} \\ 0 & if: \eta_i(q) > \eta_{0,i} \end{cases}$$

$$\text{with: } k_r > 0$$

$$f_{r,i}(q) = -\nabla_q U_{r,i}(q) = \begin{cases} k_{r,i}(\frac{1}{\eta_i(q)} - \frac{1}{\eta_{0,i}})^{\gamma-1}(-\frac{1}{\eta_i^2(q)})\nabla\eta_i(q) & if: \eta_i(q) \leq \eta_{0,i} \\ 0 & if: \eta_i(q) > \eta_{0,i} \end{cases}$$

$$U_r(q) = \sum_{i=1}^{N} U_{r,i}(q)$$

Considering $\eta_i(q)$ the clearance respect obstacle i and $\eta_{0,i}$ the *range of influence* for obstacle $i$.
The range of influence is used to tune the presence of repulsive force with respect the actual distance between the robot and the obstacle.
From this, the total force coming from the Artificial Potential Field is defined as:

$$U_t(q) = U_{a,conical}(q) + U_r(q)$$
$$f_t(q) = -\nabla_q U_t(q) = f_{a,conical}(q) + \sum_{i=1}^{N} f_{r,i}(q)$$

This total force can be used in several ways since is artificial, like a force, acceleration, or velocity. In our case, we use this as a velocity.
This method suffers of local minima points, in which repulsive and attractive force stops the robot even if is not at its destination. To avoid this problem can be used an heuristic that introduces *Vortex field forces*.
The idea is to add a repulsive action that forces the robot to go around the obstacle.

$$f_{r,i} = -\begin{bmatrix} \frac{\partial U_{r,i}}{\partial x} \\ \frac{\partial U_{r,i}}{\partial y} \end{bmatrix}$$

$$f_v = \pm \begin{bmatrix} \frac{\partial U_{r,i}}{\partial y} \\ -\frac{\partial U_{r,i}}{\partial x} \end{bmatrix}$$

$f_v$ follows the isopotential contour of the potential field.
Being this an Heuristic, the choice of the sign, and the activation/deactivation as well, can be guided from different aspects, we will discuss this implementation details in the next chapter.

## 3.4 Modules interconnection

In this brief section, we discuss the connection among the modules of the whole project.
Having HRI module and Pepper simulator running on docker image, while RA planners + House simulator running on the host machine, a solution that implies efficient communication is necessary. In the figure 3.3 we have an overall image of the scenario.
We exploited OS pipes to transfer information and instructions from the host machine's Python program to the Docker container and the opposite. With Python's os module, we specifically created pipes (FIFO) that allowed us to write data to a predetermined file path that was reachable inside the container. We were able to provide real-time inputs, configuration information, and data streams to the containerized environment using this technique.
On the host side, we developed a simple module that acts as a socket server/client, listening for incoming connections from the Docker container and sending directives for the Pepper Simulator. This script provided an interface to send commands, receive data, and manage the interaction with the container. Moreover, this socket communicates directly with the PDDL parser, when an incoming task from HRI modules is detected, consisting in a task descriptor Python dictionary.
Within the Docker container, we implemented a complementary script that behaves as a connection socket. This component established connections with the host server, enabling the Docker container to send and receive data, respond to commands, and provide feedback to the host.
In this case, the messages coming from the RA module are necessary to activate some animations from Pepper, and other similar functionalities.
In both sockets, an additional thread has been created, to allow the reception of messages in an asynchronous fashion.
This bidirectional socket communication approach allowed us to control and monitor the container's behavior, receive real-time data, and send instructions from both the host system and the Docker perspective.

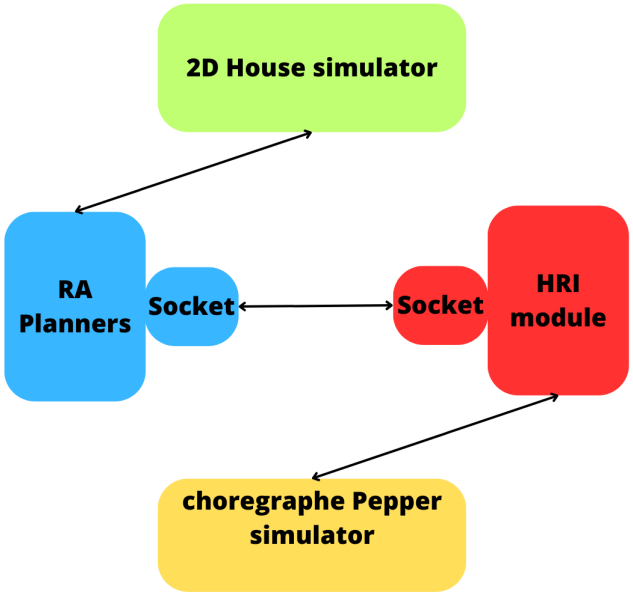


Figure 3.3: Schema representing the introduction of the project modules

# Chapter 4

# Implementation

After having described the solution proposed for each module, we move to the description of implementation details, providing the first preliminary and isolated results from each component.

## 4.1 HRI

This part of the report contains details of the HRI module that has been implemented, going through interaction aspects along with an exposition for the code snippets of interest.
The interaction is handled in a multimodal way, including vocal (here textual given the simulation), visive, through tablet and touch sensors. Moreover, is presented the extra work designed to make the robot more integrated with the task execution.

### 4.1.1 Dialog

The dialog is handled by the **ALDialog** and **ALTextToSpeech** services. Dialogue service is set to the English language, and a topic file is loaded to manage the communication with the human. The dialogue starts with a welcome from Pepper then a human message triggers the consequent response through this set of rules (figure 4.1).
Dialogue input and output are passed to callback functions that activate different functionalities:

1. Activate animations.
2. play/stop music.
3. Boot of the tablet application.
4. add/remove knowledge on the house, sending messages to RA module via socket.

This textual interaction simulates the same that would be achieved vocally using the ASR module.

```
topic: ~house-assistant()
language: enu

concept: (myname) ["My name is" "I'm" ]

proposal: %tablet Ok, Let's launch my tablet application.
        u1: ([ended] _{_*}) Ok, Do you want to do something else?
                u2: (yes) ^goto(music)
                u2: (no) ok, i stay here if you need something


proposal: %music Ok, I have a song for you, do you want to listen?
        u1: (yes) Perfect! Just a moment. Here it is. Tell me "stop" to end the music playback.
                u2: (stop) ok, i stop music.
        u1: (no) Ok, goodbye.


proposal: %conversate Do you want to conversate?
        u1: (no) Ok, goodbye.

proposal: %name What's your name?
        u1: ({~myname} _*) Hi $1

u: (["what can you do" "what you can do"]) i can help you providing different functionalities as house assistant.You can ask me to: open/close windows & doors, move objects in the house.Use the tablet to
  request task execution. Moreover you can tell me more about house objects to register new data!


u: (["can you describe the house" "tell me about this house"]) Of course! This is a beautiful house, with eight different rooms: the Foyer, the Toilet, the living Room, the dining room, the studio, the
  kitchen, and a bedroom.
Now we are in the Foyer.

u: (["task" "action" "job" "assignment"] {_*}) You can ask me to peform a task, interacting with the tablet. you want to do now?
        u1:(yes) ^goto(tablet)
        u1:(no) OK, goodbye.

u: (["tablet" "app"]  {_*}) ^goto(tablet)

u: (["hello" "hi"] {_*}) ^goto(name)

u: (["Great" "Nice" "Perfect" "thanks" "thank you"]) I'm happy to be helpful

u: (["play" "song" "music"] {_*}) ^goto(music)

u: (can i use again the tablet) Yes of course i launch my tablet application.

u: (["glasses"] {_*}) no i don't know please let me know

u: (["smartphone"] {_*}) no i don't know please let me know

u: (["glasses"] ["toilet"] ["sink"] {_*}) ok i have registered this information on the glasses

u: (["smartphone"] ["bedroom"] ["bed"] {_*}) ok i have registered this information on the smartphone.

u: (["sounds good"] {_*}) Yeah! this is a 00s anthem!

u: ([bye] {_*}) Goodbye
```

Figure 4.1: Topic file used by the dialogue manager

### 4.1.2 Tablet

Whether the Human requires the utilization of the app, the callback launches the script for the start of the server, and the HTML home page becomes interactive. The content of the web page is defined in action files, that describe the elements and what to show, as shown in figure 4.2.
In this configuration file is possible to define the user profile by changing the spoken language for the current interaction – 'it' indicates Italian, while * indicates the default value, which is English. The other three values that could be used to profile a user, together with the language, are age, gender, and occupation.
The MODIM can **execute** or **ask** (this is used when we expect output from the current page interaction) for a defined action, allowing us to configure the application in a simple way, without the necessity of code.
The pages/actions defined are the following:

1. **welcome**, a simple home page that receives the user and waits for the pressing of the button, to start the interaction.

2. **tasks**, a menu that allows to choose between 4 tasks pre-configurated, giving a brief description. Moreover, we can choose to exit and not ask for any task. (figure 4.2)

3. **navigation**, $1^{st}$ task about navigation, that commands Pepper to reach the dining room.

4. **windows**, $2^{nd}$ task about opening/closing of windows in dining room and bedroom.

5. **move known**, $3^{rd}$ task about moving cards from the table in the living room, to the table in the kitchen.

6. **move unknown**, $4^{th}$ task about moving glasses from the toilet to the studio on the desk. This task is initially unfeasible since Pepper doesn't know where the glasses are positioned, the

interaction through the dialogue system is required, to first acquire the necessary knowledge.

7. **exit**, a static view that informs the stopping of the tablet application.

in the figure 4.3 is possible to appreciate examples of these views.

```
IMAGE
<*, *, *, *>:  img/task_menu.jpg
----
TEXT
<*,*,it,*>: operazioni: 1)navigazione 2)aprire/chiudere finestre 3)spostare le carte 4)spostare gli occhiali
<*,*,*,*>:  tasks: 1)motion 2)open/close windows 3)move cards 4)move glasses
----
GESTURE
<*,*,*,*>:
----
TTS
<*,*,it,*>:
<*,*,*,*>:
----
BUTTONS
navigation
<*,*,it,*>: 1
<*,*,*,*>:  1
window
<*,*,it,*>: 2
<*,*,*,*>:  2
move_known
<*,*,it,*>: 3
<*,*,*,*>:  3
move_unknown
<*,*,it,*>: 4
<*,*,*,*>:  4
exit
<*,*,it,*>: exit
<*,*,*,*>:  exit
----
```

Figure 4.2: action file describing the task selection view

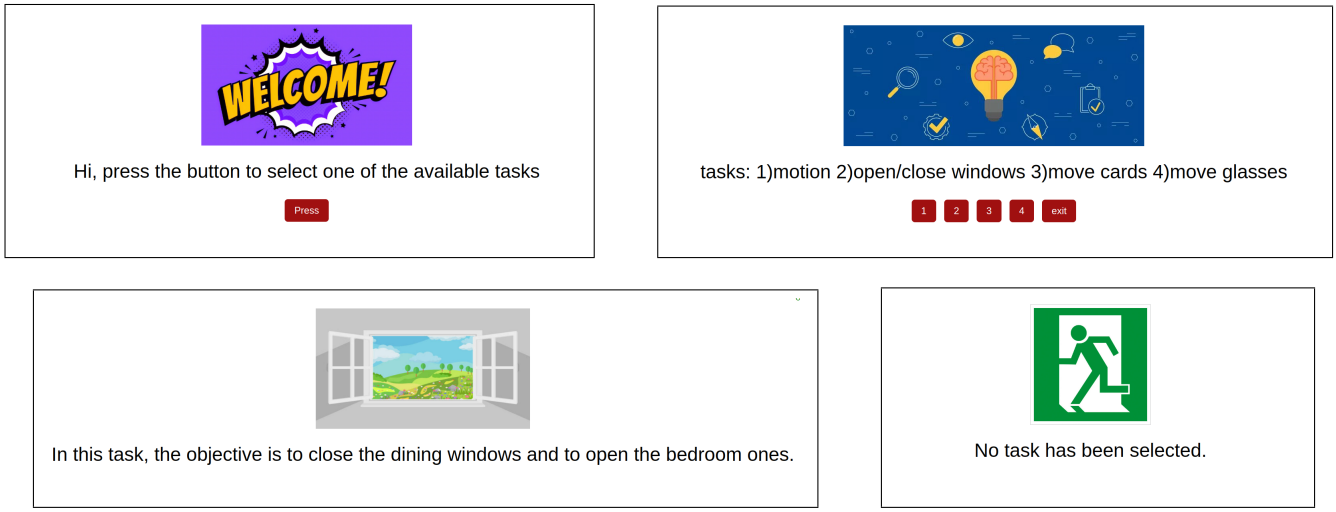


Figure 4.3: Example of tablet views: welcome page, task selection, task description and exit page

### 4.1.3 Touch

It's possible to interact with the robot, through touch sensors. The human can touch the left or right hand and the head, triggering different reactions.
Using memory service is possible to access the value of each sensor at any moment, and then the text-to-speech module sends a feedback message. In the case of Head touch, has been set a custom behavior that makes fall asleep or wake up Pepper. An example of this interaction is appreciable in the figure 4.4.

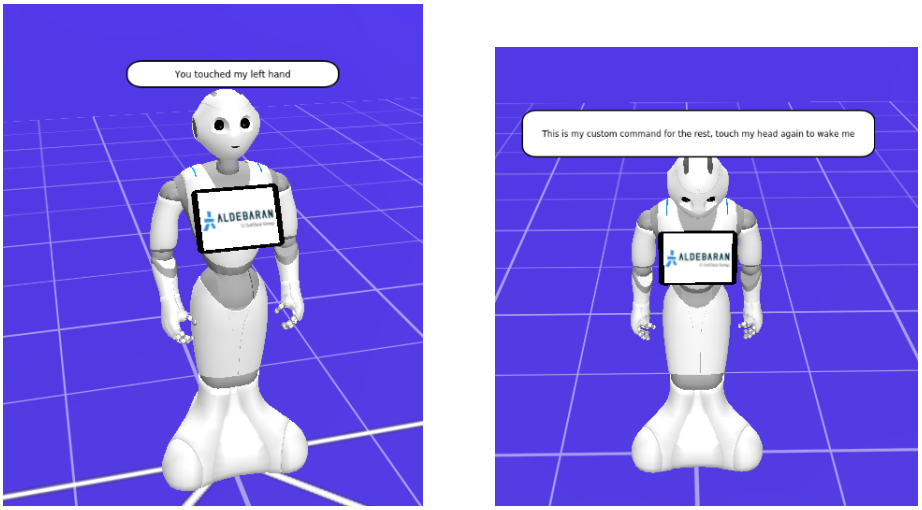


Figure 4.4: Samples from touch interaction

### 4.1.4 Extra

Different functionalities used to make more socially Pepper are explained here.
It's possible to ask for the reproduction of music from the dialogue, using the rules of the topic loaded from **ALDialog** service. This activates the **ALAudioPlayer** that permits to play audio files.
Pepper is able to reproduce different animations, defined using the **ALMotion** and **ALRobotPosture** services. The first is used to code directly the animations, using joint variables and interpolating (specifying the timing), as is shown in figure 4.5.
The second is used to call default animations like wake-up and crouch poses. Here are enlisted the custom animations defined:

- default, from any configuration the robot goes back to the initial pose.
- greet, Pepper greets to the human.
- yes/no, binary response gesture for approval or denial.
- search, Pepper looks around and scans the environment.
- grab/place, Pepper takes and drops objects.
- open/close door, to interact with house doors.
- open/close window, to interact with house windows.

These animations are activated from RA module messages, so are used contextually with the task execution. In figure 4.6 there are some captures from the execution of custom animations.

```
def grab(self): # time: 5 [s] + default = 7 [s]

    # look
    jointNames = ["HeadPitch"]
    angles = [0.3]
    times  = [1]
    isAbsolute = True
    self.motion.angleInterpolation(jointNames, angles, times, isAbsolute)


    # inflect body
    jointNames = ["KneePitch", "HipPitch"]
    angles = [-0.3, -1]
    times  = [1, 1]
    isAbsolute = True
    self.motion.angleInterpolation(jointNames, angles, times, isAbsolute)

    # move shoulder, elbow and twist while to take central object, while opening hand

    jointNames = ["RShoulderPitch","RShoulderRoll", "RElbowRoll", "RElbowYaw","RHand","RWristYaw"]
    angles = [1, -0.2,  1, 0.9, 1, 0.6]  #0.8
    times  = [1,1, 2, 2, 2, 2]
    isAbsolute = True
    self.motion.angleInterpolation(jointNames, angles, times, isAbsolute)


    # close hand
    jointNames = ["RHand"]
    angles = [0]
    times  = [1]
    isAbsolute = True
    self.motion.angleInterpolation(jointNames, angles, times, isAbsolute)

    self.open_hand = False
    self.default()
    return
```

Figure 4.5: Python code for grasp custom animation

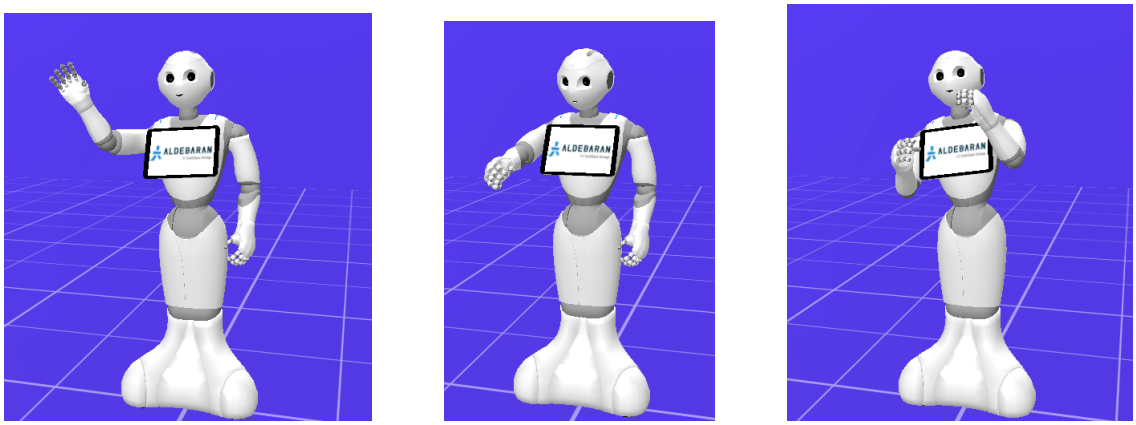

Figure 4.6: Samples from greet, open door, and dance custom animations

## 4.2 RA

In this section, we delve into the implementation details of our deterministic planning software, starting from what is explained in the previous chapter.
This module is characterized by four main parts: the PDDL templates, The PDDL parser, The Solver, and the Pepper Motion planner.

### 4.2.1 PDDL templates

As usual, PDDL templates should be at least two, one for the domain and one for the problem. The first never changes, while the second is dynamically changed by the parser.
Let's start describing the domain one first. Here we have defined types, constants, predicates, and actions as show in figure 4.7 - 4.8
Given the large number of objects present and the non-trivial nature of the problem, types are very useful and provide different advantages in the definition of these files.
The template of the problem file instead is the one present in figure 4.9. For reasons of space, the whole init section cannot be shown entirely.
How it's possible to notice, the goal section is empty and will be modified by the parser, together with the last part of the init.

```
(define (domain house_sim)
    (:requirements :strips  :adl)
    (:types
        room            - object
        direction       - object
        room_element    - object            ; superclass
        item            - room_element
        window          - room_element
        door            - room_element
        furniture       - room_element

    )

    (:constants
        north           - direction
        south           - direction
        west            - direction
        east            - direction
        free_space      - room_element    ; to characterize pepper not in proximity of room elements
    )


    (:predicates

        ; Define house environment composition
        (connected ?r1 - room ?r2 - room  ?d - direction)                  ; used to connect rooms
        (in ?o - room_element ?r - room)                                   ; for object, window, door in room
        (on ?i - item  ?f - room_element)                                  ; for object on furtniture or on another item
        (isPositioned ?o - room_element ?r - room ?d - direction)        ;generic predicate for define the cardinal position of windows and doors.


        ;Interaction with doors and windows  (closed wolrd assumption for closed elements)
        (openDoor  ?w - door)
        (openWin   ?w   - window)


        ;Pepper information
        (PepperIn  ?r - room)
        (PepperAt  ?o - room_element)                                    ;generic to indicate pepper near to window/door/furniture (now can interact)
        (PepperHas ?i - item)
        (freeHands)
    )
```

Figure 4.7: PDDL domain, types and predicates

```
    ;                               [motion action]
    (:action move2
        :parameters (?r - room ?from ?to -room_element)
        :precondition (and (PepperIn ?r) (PepperAt ?from) (in ?from ?r) (in ?to ?r))
        :effect (and (PepperAt ?to) (not(PepperAt ?from)))
    )

    (:action move2room
        :parameters (?from ?to - room  ?d - door ?side - direction)
        :precondition (and (connected ?from ?to ?side) (isPositioned ?d ?from ?side) (PepperIn ?from) (PepperAt ?d) (openDoor ?d) (in ?d ?from)(in ?d ?to))
        :effect (and (not(PepperIn ?from)) (PepperIn ?to))
    )

    ;                               [action with windows and doors]
    (:action open_door
        :parameters (?e - door ?r -room)
        :precondition (and (not(openDoor ?e)) (in ?e ?r) (freeHands) (PepperIn ?r) (PepperAt ?e))
        :effect (and (openDoor ?e))
    )

    (:action close_door
        :parameters (?e - door ?r -room)
        :precondition (and (openDoor ?e) (in ?e ?r) (freeHands) (PepperIn ?r) (PepperAt ?e))
        :effect (and (not (openDoor ?e)))
    )

    (:action open_win
        :parameters (?e - window ?r -room)
        :precondition (and (not(openWin ?e)) (in ?e ?r) (freeHands) (PepperIn ?r) (PepperAt ?e))
        :effect (and (openWin ?e))
    )

    (:action close_win
        :parameters (?e - window ?r -room)
        :precondition (and (openWin ?e) (in ?e ?r) (freeHands) (PepperIn ?r) (PepperAt ?e))
        :effect (and (not (openWin ?e)))
    )

    ;                               [action with house objects]

    (:action grab_object
        :parameters (?i - item ?r - room ?f - room_element)
        :precondition (and (in ?i ?r) (in ?f ?r) (on ?i ?f) (freeHands) (PepperAt ?f) (PepperIn ?r))
        :effect (and (not(freeHands)) (not(on ?i ?f)) (not(in ?i ?r)) (PepperHas ?i))
    )

    (:action place_object
        :parameters (?i - item ?r - room ?f - furniture)
        :precondition (and (in ?f ?r) (PepperHas ?i) (PepperIn ?r) (PepperAt ?f) )
        :effect (and (not (PepperHas ?i)) (freeHands) (in ?i ?r) (on ?i ?f))
    )
)
```

Figure 4.8: PDDL domain, actions

```
(define (problem house-motion)
    (:domain house_sim)
    (:objects
        foyer living_room dining toilet studio bedroom kitchen outdoor          - room
        d_foyer_outdoor d_foyer_living  d_toilet_living d_studio_living         - door
        d_bedroom_living d_living_dining d_dining_kitchen                       - door
        wl_foyer  wl_toilet wl_studio wl_bedroom wl_dining wl_kitchen           - window
        wr_foyer  wr_toilet wr_studio wr_bedroom wr_dining wr_kitchen           - window
        green_marker pen pencil plate_empty cup_coffee plate_oranges            - item
        plate_apples orange1 orange2 orange3 apple1 apple2 smartphone           - item
        red_notebook green_notebook glasses yellow_notebook cards pink_notebook - item
        desk_studio pool_studio kitchenette table_kitchen bed cabinet_bedroom_l - furniture       ;_l -> left _r -> right
        cabinet_bedroom_r tv_bedroom water tub sink cabinet_toilet              - furniture
        tv_living sofa table_living armchair_l armchair_r table_dining          - furniture
    )

    (:init

        ;                           room connections
        (connected outdoor foyer  north)
        (connected foyer outdoor  south)
        (connected foyer living_room  west)
        (connected living_room foyer  east)
        (connected toilet living_room  south)
```

```
        (on red_notebook cabinet_bedroom_l)
        (on green_notebook cabinet_bedroom_r)
        (on glasses sink)
        (on yellow_notebook cabinet_toilet)
        (on cards table_living)
        (on pink_notebook table_dining)

        ;                           pepper init
        (PepperIn foyer)
        (PepperAt free_space)
        (freeHands)
    )

    (:goal
        (and )
    )
)
```

Figure 4.9: PDDL problem tamplate

### 4.2.2 The Parser

The main difficulty of defining the planning problem with PDDL files is the static nature of the data. In our case, the problem file should be modified based on the type of task required. Moreover, we want also to do tasks in sequence, that's why the problem file should be updated with the new states (adding and removing predicates), including a kind of memory.
For these reasons has been developed a Parser software, which is responsible for generating the parsed problem file, starting from the template presented in the previous section.
An example of the final part of the problem file, modified by the Parser is in figure 4.10.
The parser has a complete knowledge of the formalism used for the predicates and actions in the templates. The goal is generated starting from a python dictionary that describes the task with a fixed format, which is then traduced in predicates, and appended to the file.
The init is modified by analyzing the previous plan executed, knowing the effects of each action. In this way the init predicates can be removed, and inserted at the end of the section.

Thanks to this software is also possible, besides what regards execution of plans, to remove and insert initial knowledge, using the same criterion of the init update.

```
181         ;                           pepper init
182         (openDoor d_bedroom_living)
183         (openWin wl_bedroom)
184         (openWin wr_bedroom)
185         (openDoor d_dining_kitchen)
186         (freeHands)
187
188         (on cards table_kitchen)
189         (PepperIn living_room)
190         (PepperAt free_space)
191         (in cards kitchen)
192     )
193
194     (:goal
195         (and (on glasses desk_studio) (PepperIn studio) (PepperAt free_space)  (freeHands))
196     )
197 )
```

Figure 4.10: Final part of PDDL problem file from the Parser

### 4.2.3 The Solver

After the generation of coherent PDDL files, is necessary to convert this in a Plan. The solver is responsible for this operation.
It simply performs a system call to the *Metric FF* deterministic planner. The terminal output is then filtered, organizing the plan into a convenient list of actions.

### 4.2.4 Pepper motion

Based on the theoretical notions previously exposed, we can describe the implementation, with a presentation of the developing choices about APF method.
The first crucial aspect in order to define the APF is the computation of the clearance, used for the repulsive and and vortex force. In this case, a slight modification of the algorithm has been applied. Instead of using a clearance value for each obstacle, the APF module computes one single value of clearance, the lower one. The computation of clearance is performed separately for the walls and obstacles using Euclidean distance, and then the smaller value is taken.
If not specified in the house simulator prompt, this minimum clearance is visualized with the red dot on the map. This modification implies then a single repulsive force.
The specifics of attractive and repulsive force are exposed in the following table:

| **Parameter** | **Value** |
|---|---|
| $K_a$ | 0.05 |
| $K_r$ | 700 |
| $\gamma$ | 2 |
| $\eta_0$ | 20 |

Also the vortex field heuristic implementation has been customized, in order to have a coherent choice for the sign of the force (always orthogonal to the repulsive force). The pseudocode is presented below (algorithm 1).

$\epsilon$ is a new parameter, to choose the vortex force direction. The total force is the sum of attractive, repulsive, and vortex forces.
Finally, a normalization of the total force is performed in order to always stay bounded on the maximum feasible velocity. An example of execution of this technique is presented in the figure 4.11

**Algorithm 1** Vortex force computation

**Require:** repulsive force $f_r$, attractive force $f_a$
**Ensure:** vortex force $f_v$
  $\epsilon \leftarrow 0.1$
  **if** $||f_r|| == (0, 0)$ **then**
    remove saved $f_v$
    $f_v \leftarrow (0, 0)$
  **else**
    **if** not exists saved $f_v$ **then**
      $f_{v,-} \leftarrow$ rotate_cc($f_v$)
      $f_{v,+} \leftarrow$ rotate_ccw($f_v$)
      $e_- \leftarrow ||f_a - f_{v,-}||$
      $e_+ \leftarrow ||f_a - f_{v,+}||$
      $\delta \leftarrow e_- - e_+$
      **if** $\delta > \epsilon$ **then**
        $f_v \leftarrow f_{v,+}$
      **else if** $\delta < -\epsilon$ **then**
        $f_v \leftarrow f_{v,-}$
      **else**
        $f_v \leftarrow$ closer_to_center_room($f_{v,-}$,$f_{v,+}$) {Return force closer to room center.}
        save $f_v$
      **end if**
    **else**
      $f_v \leftarrow$ saved $f_v$
    **end if**
  **end if**
  **return** $f_v$

## 4.3 House simulator

Thanks to Pygame's graphical capabilities to create a visually appealing and interactive 2D environment, we have constructed an environment that represents the house and its surroundings, providing a realistic (simplified for the two dimensions) setting for simulating robot motion.
For The development of this simulator, massive work has been carried out:

- Coding of the main loop rendering, handling PyGame events, coming from the interaction with the simulator (buttons and command prompt).
- Creation and collection of assets like: textures, furniture and objects images, sounds, and font.
- definition of display object classes, representing UI (input/output boxes, buttons, Texts) and the environment (Rooms, Doors, Windows, Furniture and Pepper).
- Instantiation of the environment, creating and placing display objects for the house representation and UI as well.

The overall aspects of the simulator can be seen in the figure 4.12.

Figure 4.11: The custom APF method in different situations. The red dot represents the clearance, while the green dot is the actual target position

Besides the input functionalities coming from the buttons in the UI, it's possible to use the prompt to activate different functionalities and tests thanks to the input interpreter.

- **reset**, reset the simulation environment.
- **debug**, toggle the debug mode, useful to define the position of display objects in the environment.
- **obs**, toggle the obstacle mode visible in figure 4.13. The obstacles are marked in red, while movable objects in blue.
- **clearance**, show clearance of the nearest obstacle.
- **target**, show the target of the actual motion with a point on the map.
- **direction**, show a single vector representing the pepper motion.
- **forces**, show all the force coming from the APF method that generates the final motion direction.
- **test_***, different test commands used during the development to test the simulator.

Display objects have several properties that are interesting to highlight. Doors and windows (left and right windows are handled independently) can be opened and closed by calling a method. Furniture

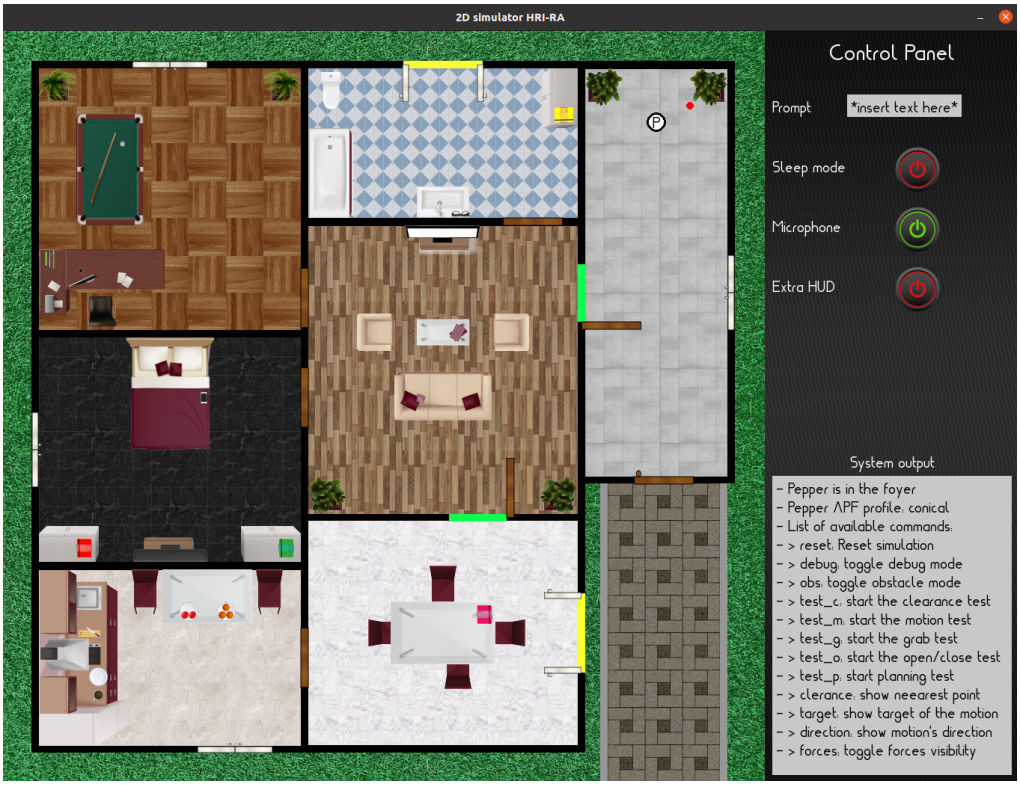


Figure 4.12: House simulator

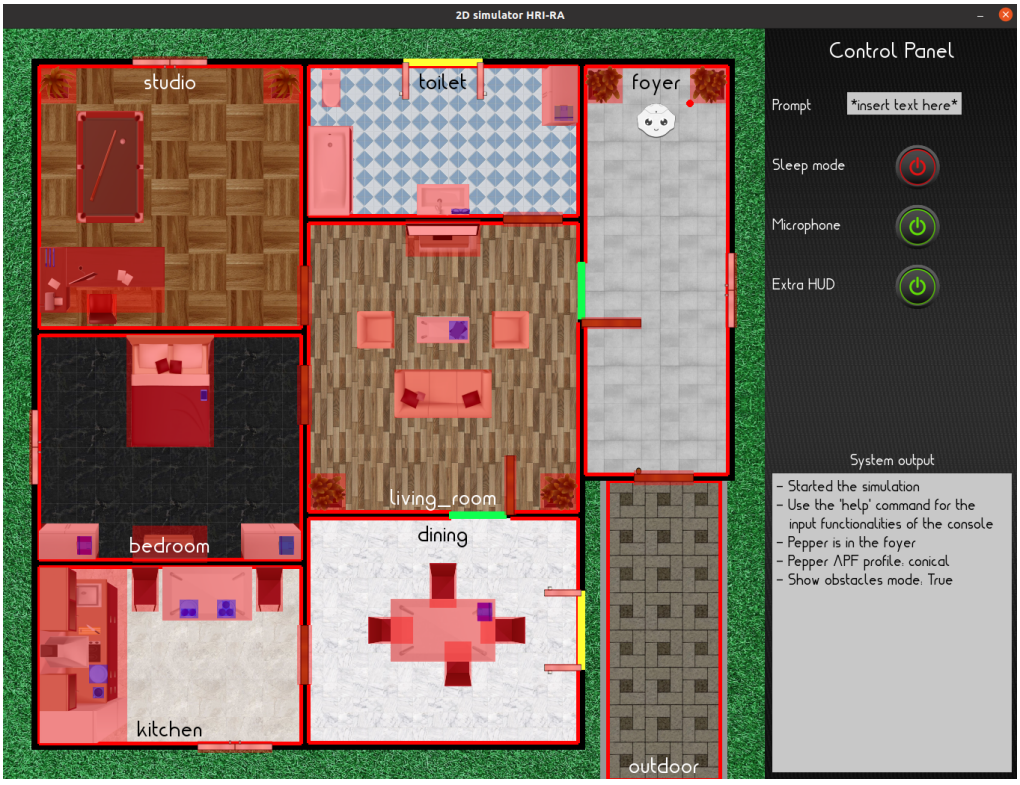


Figure 4.13: House simulator with extra HUD and show obstacle mode on

and Objects differ since the first are fixed in the scenario, while the seconds can be moved by Pepper. The Pepper display class includes the functionalities coming from the RA module. Providing the plan, it decomposes its actions, calling internal methods for the execution.
This class offers methods for each possible motion: to another room, to furniture, to a window, etc. Together with methods to open and close doors/windows, place and grasp objects. It uses the APF method from RA module for the motion inside a Room.
As described in the figure 3.3, this software interacts directly with the RA module, which provides and updates the pepper placeholder position.

# Chapter 5

# Results

To have a better idea of the final results is suggested to give a look at the demo presentation that has been recorded, which is publicly accessible at this link.
In it, is remarked the many stages of the interaction, and task execution, showing the interconnection between all the modules designed.
Figure 5.1 shows the collaboration and interplay of the developed software, which provides results on the two simulators used for this project.
Here from Choregraphe has been extracted only the robot and dialog view.

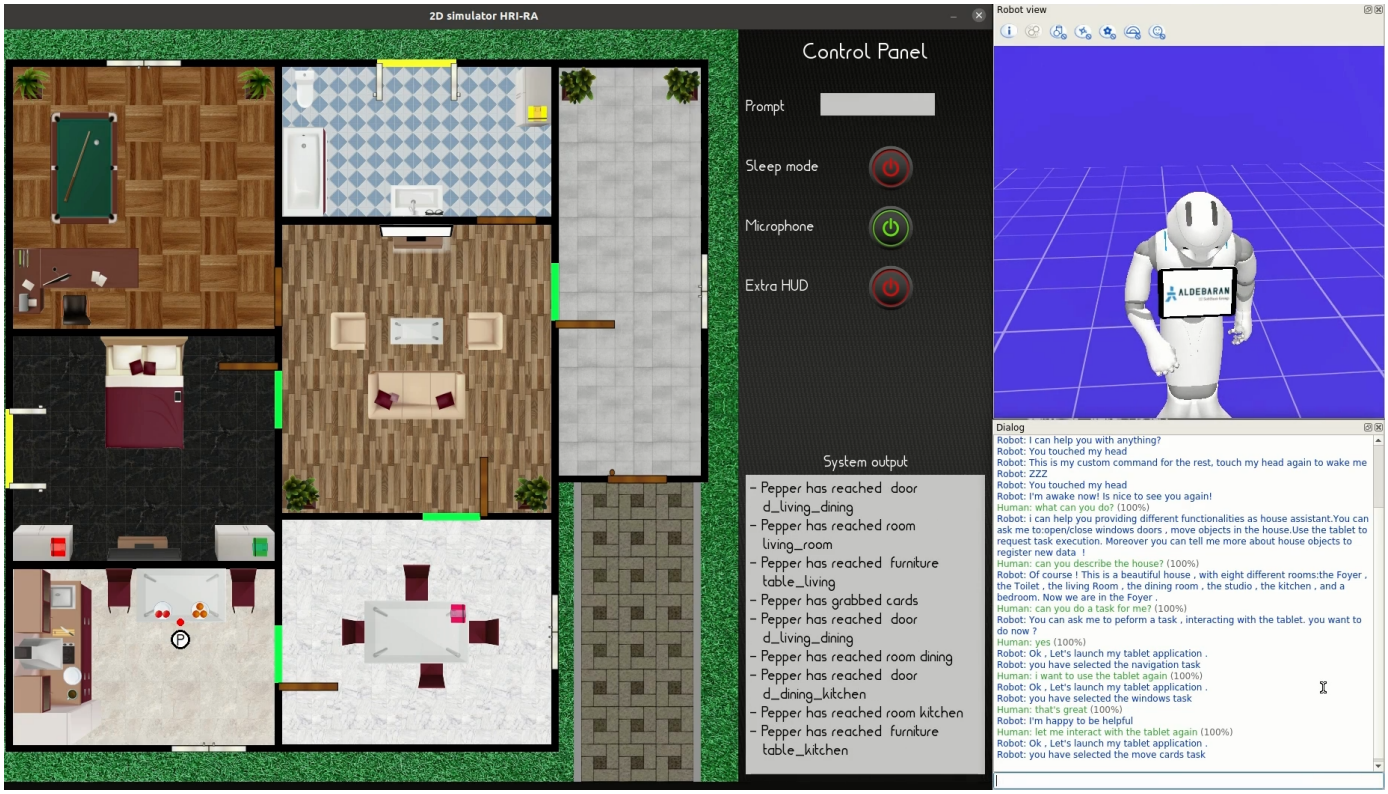


Figure 5.1: Capture of the final demo, while Pepper is placing an object

# Chapter 6

# Conclusion

The use of robots and artificial intelligence in domestic settings has experienced a significant transition in recent years. Robotics integration into homes is now both a societal necessity and a reality in terms of technology. Robotic home aid has transformed from a sci-fi idea to a necessary component of contemporary living.
In this study, we have presented the development of a system that integrated the potentialities of Human-robot interaction and Reasoning Agents. Introducing different solutions and techniques to handle the study case of Pepper robot as house assistant.
Considering the simulation nature of this project, we can be satisfied with the contribution obtained from this work, even if it's important to always keep in mind the limitations with respect to a realistic scenario.
This project allowed me to work in a very stimulating and fascinating context, learning how to use different tools that collaborate all together to reach a final goal.

## 6.1 Future directions

We conclude by presenting some possible ways to continue this work and improve the system presented.

- Pass from a 2D house simulator to a 3D version, capturing image samples that can be used directly from the Pepper image recognition system.
- Improvement of Vortex field heuristic for harder cases.
- Substitute the deterministic planner with a probabilistic one, modeling the problem in a more realistic manner.
- Upgrade the number of tasks realizable from the robot.
- Upgrade Table application, in order to select in a custom manner the task to perform. Note that the RA module is already compatible with this modification, being able to perform any task given the domain exposed.

# Bibliography


[1] *A survey of socially interactive robots*, Fong, Terrence, Illah Nourbakhsh, and Kerstin Dautenhahn. Robotics and autonomous systems 42.3-4 (2003), pp. 143-166

[2] *The Uncanny Valley [From the Field]* M. Mori, K. F. MacDorman and N. Kageki, IEEE Robotics & Automation Magazine vol. 19, no. 2, pp. 98-100, June 2012, doi: 10.1109/MRA.2012.2192811

[3] *What is a robot companion - friend, assistant or butler?* K. Dautenhahn, S. Woods, C. Kaouri, M. L. Walters, Kheng Lee Koay and I. Werry IEEE/RSJ International Conference on Intelligent Robots and Systems, Edmonton, AB, Canada, 2005, pp. 1192-1197, doi: 10.1109/IROS.2005.1545189.

[4] *Robotic home assistant Care-O-bot® 3 - product vision and innovation platform*, B. Graf, U. Reiser, M. Hägele, K. Mauz and P. Klein, IEEE Workshop on Advanced Robotics and its Social Impacts, Tokyo, Japan 2009, pp. 139-144, doi: 10.1109/ARSO.2009.5587059.

[5] *Use of social commitment robots in the care of elderly people with dementia: A literature review*, Mordoch, Elaine, et al., Maturitas 74.1 (2013), pp. 14-20.

[6] *Pamini: A framework for assembling mixed-initiative human-robot interaction from generic interaction patterns* Peltason, Julia and Wrede, Britta, Proceedings of the SIGDIAL 2010 Conference, pp. 229–232, 2010

[7] *Finite LTL synthesis as planning* Camacho, Alberto and Baier, Jorge and Muise, Christian and McIlraith, Sheila, Proceedings of the International Conference on Automated Planning and Scheduling, vol. 28, pp. 29–38, 2018,

[8] *A Numeric PDDL Based Approach for Temporally Constrained Journey Problems*, S. C. M. Caff, F. Di Mauro and E. Scala, IEEE 26th International Conference on Tools with Artificial Intelligence, Limassol, Cyprus, 2014, pp. 99-106, doi: 10.1109/ICTAI.2014.25.

[9] *The FF planning system: Fast plan generation through heuristic search.* Hoffmann, Jörg, and Bernhard Nebel. Journal of Artificial Intelligence Research 14 (2001), pp. 253-302.

[10] *The Metric-FF Planning System: Translating "Ignoring Delete Lists" to Numeric State Variables.* Hoffmann, Jörg, Journal of artificial intelligence research 20 (2003), pp. 291-341.