# MARIO: Pepper assistant for patients with senile dementia

*Human-Robot Interaction*

*Sveva Pepe and Simone Tedeschi*


*Master's Degree in Artificial Intelligence and Robotics*

*Sapienza University of Rome*

*July 2021*

# Contents

**1 Introduction** **1**

**2 Related work** **2**

**3 Solution** **3**

**4 Implementation** **5**
4.1 Human Detection . . . . . 5
4.2 Welcome . . . . . 6
4.3 Tablet . . . . . 8
4.4 Music . . . . . 13
4.5 Vision . . . . . 17
4.6 Touch . . . . . 19

**5 Results** **20**

**6 Conclusions** **20**

*The authors equally contributed to the project.*

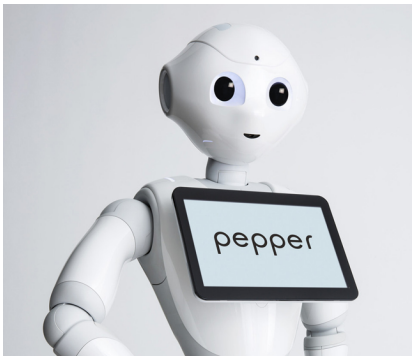
# 1 Introduction

Nowadays, the unceasing search of the progress in the Artificial Intelligence field aims to improve the human condition, making changes to the society in which we live and, in particular, in the case of Robotics field, aiming at the inclusion of assistant robots into everyday life. Assistant robots perform tasks useful to the welfare of humans in an autonomous way. Recent studies have demonstrated that their inclusion could in the future bring great improvements to our society, helping in those sectors where today there are still difficulties, like hospitals [3]. In fact, lately, it has been noticed how robots can be of great support to elderly person [2], who need continuous assistance, which cannot always be satisfied by the hospital staff. In fact, in this work we will see MARIO, a Pepper robot (see Figure 1a) interacting with a person suffering from senile dementia, one of the most delicate pathologies. The aim of this project consists of assisting, socializing and keeping company to these people, generally elderly, who need constant attention, and their health could benefit from these interactions, letting them not feeling alone and neglected.

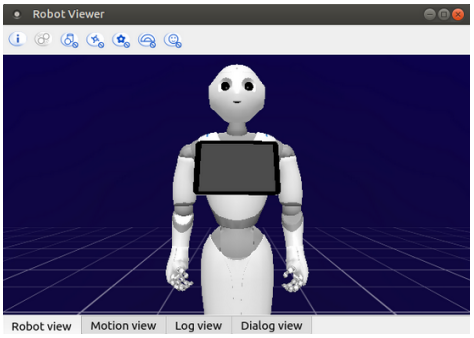
In this work we tried to focus on several aspects of the Human-Robot Interaction (HRI) field and its importance in healthcare environments, especially when an assistant robot would be immersed in an environment with the aim to contribute to the improvement of the hospital services, bringing huge help to the whole medical staff and becoming part of it. In our case, MARIO must be able to propose and carry out rather simple tasks since patients with senile dementia tend to forget things, including the simplest ones.

Our idea is to place MARIO in an hospital, or a nursing home, where there are people with senile dementia. People can interact with the robot through the tablet (placed on the robot's chest) or it is possible to directly conversate with it. MARIO provides information about news and weather forecasts, allows you to listen songs and even test your memory with a very simple game. Then, to let MARIO be more social and sustain a more natural interaction, we implemented several gestures. As an example, when reproducing a song of a specific musical genre, MARIO moves accordingly to the reproduced song to entertain the patient, letting the interaction be more enjoyable. We have also introduced a computer vision system that enables MARIO to understand the patient's emotions, and we employed this system to monitor them when they are listening to a song, in order to understand if that particular song is liked or not. So, the robot is able to understand the situations and give advice according to the patient's behavior. Moreover, MARIO, through its sonar, can perceive the humans in its vicinity, and this capability lets MARIO approach patients. Finally, the robot can also perceive if a person touched him.

Our project was developed mainly in Python using NAOqi[1], a SDK from Softbank Robotics, to connect directly to the "robot", which provides the needed APIs.




(a) Pepper Robot




(b) SDK for Android emulator

[1] `http://doc.aldebaran.com/2-1/index.html`

We also used MODIM (Multi-modal Interaction Manager) to start the server that we use for interacting with the tablet. Given the covid situation we could not work on the physical robot, hence we used the SDK for Android emulator, see Figure 21b, because it allowed us to analyze the behavior of the robot in an almost real way based on our instructions.

Summarizing, the benefits of having an assistant robot like the one we propose in our project are:

- increase the mental health of the patients by providing constant social contacts and stimulating interactions;
- visit patients and take care of them when the hospital has a lack of staff;
- increasing the profit of the hospital.

# 2 Related work

This Section concerns relevant previous works and scientific publications in HRI field which have influenced the overall project, transmitting us ideas and different points of view. In particular, we will show papers on human-robot interactions in the healthcare system.

In recent years there has been a big increase in the field of robotics. Robots are entering many sectors and, above all, they are starting to enter the healthcare sector. However, as the aging population continues to grow, the current medical staff and healthcare workers are increasingly burdened with treating the ever increasing number of elderly patients, in particular, those with degenerative diseases such as senile dementia, Alzheimer's, etc. As a result, we can expect to see big improvements in the field of medical robotics, especially for that sub-category of robots that serve to provide social facilitation and companionship to users, which are called Socially Assistive Robots (SARs).

SARs are the union of Assistive Robots and Socially Interactive robots, with the aim of providing assistance to human users through social interaction. Attempts to develop SARs for interaction with dementia patients are not uncommon. The so-called social robots have been developed for several years, some of which take the form of relatively simple pseudo-animal, such as NeCoRo, AIBO and Paro, which represent a cat, Greenland dog and seal, respectively. These robots have been experimented in nursing homes for the elderly [8]. Then, there is "Bandit", an anthropomorphic robot that represents the human torso and upper body with more advanced social tails, such as speech and interaction through a simple game [1]. Additionally, such devices typically have roles in helping patients to keep a schedule and remember certain activities. More complicated models use computer vision to determine what the user is trying to do and urge them to carry out the task correctly. Even more complex social robots, for example, NAO robots, have been employed to engage patients in various types of therapy, i.e. language, music and physiotherapy, as well as storytelling sessions. Simplified interactions have been implemented for the use of such robots with severe dementia patients, as they are not completely able to keep attention in structured sessions, highlighting the need for different development strategies depending on on the severity of dementia [7]. Above all, it is important that a SAR is able to recognize emotions and behave accordingly. A robot that can measure human emotions means that it is able to detect sadness or physical pain and alert doctors. Furthermore, Dementia patients have been shown to respond positively to socio-emotional interactions with a robot [5]. While on the surface the emotion recognition task seems to be a quite complex challenge, there have been several advances in computer vision that can make this task solvable in relatively easy ways. In particular, thanks to recent advances with Deep Neural Networks (DNNs), the family of tasks about image recognition has become relatively trivial to solve [6], and the accuracy of such systems keeps to increase with new architectures and pretrained models [4]. In our work we have tried to draw inspiration from all these works in order to create a SAR as general as possible, which has both simple and clear interactions to the patient, and slightly more complex interactions. Further details will be provided in Section 4.

In particular, our goal was to make the robot as social as possible to ensure that patients do not see it as a stranger, but rather that they are able to benefit from its presence and that they can think one day, to accept its integration into their own daily lives.

# 3 Solution

Our work has been developed entirely in the Linux environment where, thanks to Docker, it has been possible to create one or more independent containers in which is possible to work independently without modifying the entire system. It is made possible by providing an additional abstraction through virtualization at the operating system level by executing the following commands: i) run a docker image with all the considered libraries:

```
./run.bash
```

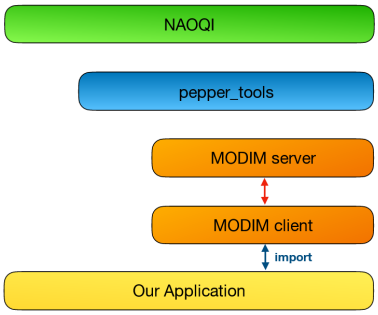
and run such image to enter the architecture of your robot:

```
docker exec -it pepperhri tmux
```

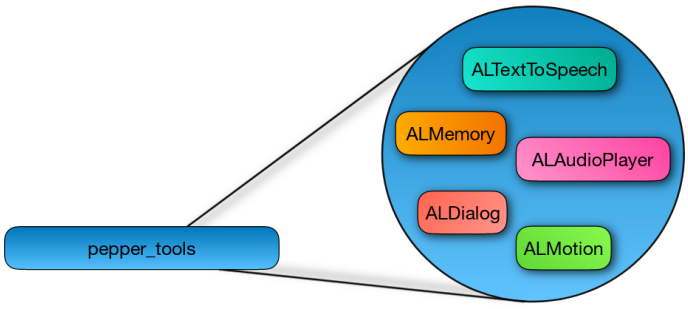
As above mentioned, our project was built in Python, which is used to manage the libraries made available by NAOqi, but also to communicate with the HTML file, used for the management of the tablet.

As seen in Section 1, we use the Pepper robot, which uses the NAOqi operating system, or an embedded Linux distribution developed specifically to meet the needs of the robots of Softbank Robotics. It provides several programs and libraries, the NAOqi APIs, allowing the robot to come to life through, for example, dialogue, gestures, robot movements etc. This is the basis on top of which our software architecture is built, see Figure 2a. To communicate with the robot we need to launch the NAOqi server, and in our case it is activated when we open the SDK for android emulator. Then, thanks to the available tools – the *pepper_tools* – and the documentation of Softbank, we can implement the whole interaction with the robot. For example, we can use such tools to move the robot in the direction of the person to start a conversation. In particular, in Figure 2b, the pepper_tools used in our work are shown. Let's briefly analyze each of them in detail.

The `ALTextToSpeech` is a module that allows the robot to speak. It sends commands to a text-to-speech engine, and also authorizes voice customization. The result of the synthesis is then sent to the robot's loudspeakers. `ALMemory` is a centralized memory used to store all key information related to the hardware configuration of the robot. More specifically, it provides information about the current state of the Actuators and the Sensors. In addition, this module can also be used to store and retrieve named values, and act as a hub for the distribution of event notifications. `ALAudioPlayer` provides playback services for multiple audio file format and the associated common functionalities (play, pause, stop and loop, among others). In all cases, the resulting audio stream is sent to the robot's loudspeakers. `ALDialog` is a module that allows you to endow the robot with conversational skills by using a list of "rules" written and categorized in an appropriate way. `ALMotion` module provides methods which facilitate the robot movements. In particular, it contains four major groups of methods for controlling: i) joint stiffness (basically motor On-Off), ii) joint position (interpolation, reactive control), iii) walk (distance and velocity control, world position and so on), and iv) robot end effector in the Cartesian space (inverse kinematics, whole body constraints).




(a) Architecture of our software




(b) pepper_tools in details

We have also employed MODIM, a service that works thanks to the definition of some interaction functions which can be easily invoked. There are two ways for using MODIM:

1. "direct call", where each function can be directly called using the `im.executeModality(MODALITY, INTERACTION)` command, specifying the required interaction and what it wants to perform;

2. "call actions", where with simple commands like `im.execute(ACTION)` or `im.ask(ACTION)` we can work indirectly with "actions" implemented separately, each in a different file. An example is shown in the Figure 3.

In particular, the modalities available are: i) `TTS` (Text To Speech) that allows the robot to speak, ii) `ASR` (Automatic Speech Recognition) that allows voice interaction with the robot, iii) `TEXT_DEFAULT` to write text on the tablet touch screen, iv) `TEXT_TITLE` to assign a title to the current modality and show it on the tablet touch screen, v) `BUTTONS` to generate buttons on the touch screen layout, and vi) `IMAGE` to load any needed image on the tablet screen.

We can also see from Figure 3, that in the action files it is also 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 – and this could be a very important feature if we put Pepper, for example, in a multicultural environment. The other three values that could be used to profile a user, together with the language, are age, gender and occupation.

```
IMAGE
<*, *, *, *>:  img/activity.jpeg
----
TEXT
<*,*,it,*>: Cosa ti piacerebbe fare?
<*,*,*,*>: What would you like to do?
----
TTS
<*,*,it,*>: Cosa ti piacerebbe fare?
<*,*,*,*>: What would you like to do?
----
BUTTONS
news
<*,*,it,*>: Leggere le notizie
<*,*,*,*>:  Read the news
meteo
<*,*,it,*>: Previsioni meteo di oggi
<*,*,*,*>:  Forecast of today
game
<*,*,it,*>: Gioca ad un gioco
<*,*,*,*>:  Play a game
exit
<*,*,it,*>: Indietro
<*,*,*,*>:  Exit
----
```

Figure 3: Example of action file

In our project we included only the second method, being more readable and tidy. Moreover, MODIM allows to render on our local web page – through the `im.displayLoadURL()` command – all what we have implemented on its "touch screen", thus having a visual feedback of what just done. However to do this work we needed another tool: a web server that connects our localhost HTML page with our architecture. So, it is here that the importance of the HTML code developed comes out, in which you can define the layout of the Pepper tablet according to your liking.

Finally, to let the NAOqi server and MODIM work properly, we need to execute the following steps:

- To activate the NAOqi server start the emulator: go to Android Studio > Tools > Pepper SDK > Emulator;

- open the html file $HOME/playground/Pepper-Interaction/project-pepper/tablet/index.html on the browser
- To enable the MODIM server, go to $HOME/src/modim/src/GUI and run python ws_server.py

# 4 Implementation

In this Section we provide a detailed explanation of the implementation of our project, describing step-by-step all the possible interactions and the corresponding code. In particular, as we will see in a moment, we made the interaction as multimodal as possible, handling it through 5 communication channels – written, vocal, tablet, visual and touch interactions – and provided further tools to make the robot even more socially acceptable.

## 4.1 Human Detection

We equipped our robot MARIO with a (fictitious) sonar-based system for detecting humans. In particular, before starting the interaction, MARIO identifies all the humans in the room (for simplicity we assumed that the robot can perceive only humans in front of it), and computes all the corresponding distances. In the meanwhile, MARIO performs a gesture simulating the movement of searching for someone by rotating its head left and right, see Figure 4. Details about how animations are created will be given in Section 4.4.

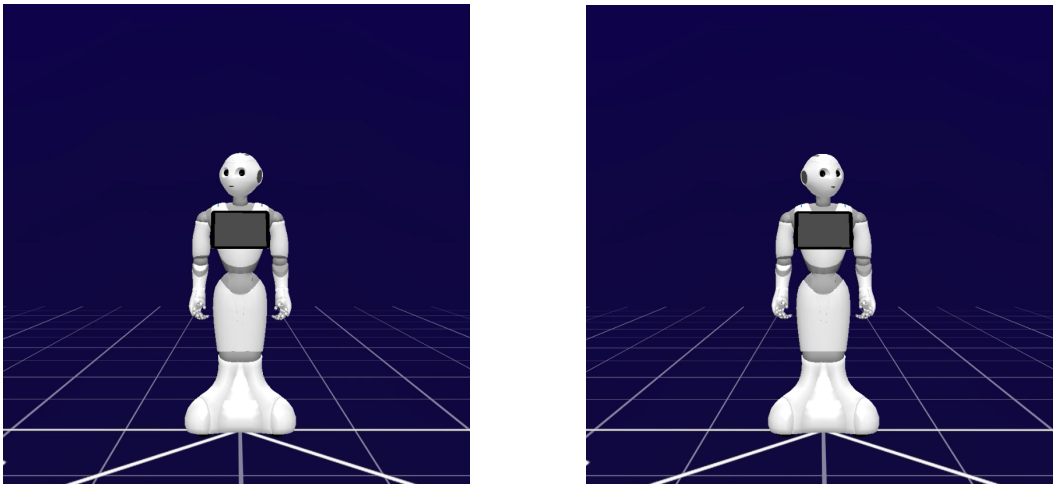


Figure 4: Searching gesture

Then, once all the distances have been collected, MARIO chooses the minimal one and move towards the closest person. To achieve this behavior we use the following fragment of code.

```
1     # Sonar
2     sonar = Sonar(ALMemory, robot_position)
3     sonar.set_sonar()
4
5     # Gesture
6     gesture.gestureSearching()
7
8     # Motion
9     motion = Motion(ALMotion)
10    distances = sonar.get_distances()
11    min_distance, idx = motion.selectMinDistance(distances)
12    motion.forward(sonar, min_distance)
13    sonar.robot_position = tuple(map(operator.sub, sonar.humans_positions[idx],(0.5,
      0)))
```

More specifically, in the second line, when we initialize the sonar providing as input the initial robot position, the system internally calls a function to retrieve all the positions of the humans in

the room (their positions are randomly set). Then, in the third line we align the values of the sonar sensors with the actual ones of the simulated robot. In the sixth line instead, we start the gesture animation. In the rest of the code the robot computes all the distances of the humans with respect to the current robot position, it selects the minimal one and it moves toward the corresponding person keeping a distance of 0.5 meters by the human.

## 4.2 Welcome

Once MARIO identifies the person with which it wants to interact, the interaction starts with a welcome message, launched using the TTS modality. In the meanwhile, in an asynchronous way, MARIO performs an "Hello" gesture, moving its right arm, wrist and hand (see Figure 5). Details about all the animations will be given in Section 4.4.

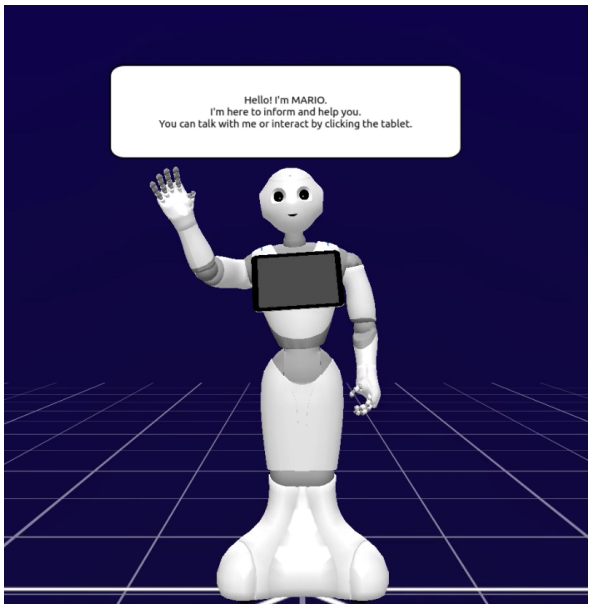



Figure 5: Hello gesture

The code below shows what just described.

```
1    tts_service.say("Hello, I'm MARIO. I'm here to inform and help you. You can talk
     with me or interact by clicking the tablet." + " "*5, \_async=True)
2    gesture.doHello()
```

From now on the human can interact with the robot by writing (in a simulated environment only writing is allowed, but with a real robot it is also possible to speak) in the Dialog view made available by the emulator. The dialogues are managed through `.top` files, which represent the interaction topics. For instance, we make use of a `main.top` file which enable to perform all the activities mentioned in Section 1. A fragment of such file is reported below:

```
1    topic: ~assistant()
2    language: enu
3
4    u:(["hi" "hello"] {_*}) ^goto(name)
5
6    proposal: %name What's your name?
7      u1: ({~myname} _*) Hi $1
8
9    concept: (myname) ["My name is" "I'm"]
10   ...
```

For example, when the human will answer to MARIO with a command of the form "Hi", "Hello", "Hello MARIO" or "Hi Pepper", such command will be identified through the fourth line of the `main.top` file, where MARIO then asks for the name of the human going to the proposal at line 6.

At this point, depending on the name of the user, MARIO will behave in a certain manner. The user name is taken based on the last answer of MARIO, the one on the line 7 of the `main.top` file and then, MARIO checks in the database if it already met that user, and if this is the case then the robot will say "Welcome back", otherwise it will insert the new user in the database and says "Nice to meet you". The following code shows what we have just described.

```
    lastAnswer = ALMemory.subscriber("Dialog/LastAnswer")
    lastAnswer.signal.connect(handleLastAnswer)

    def handleLastAnswer(lastAnswer):
        ...
        elif "Hi" in lastAnswer:
            name = lastAnswer.split()[1].lower()
            if name.lower() in database.patients:
                database.addPatient(name.lower())
                tts_service.say("Nice to meet you! Do you want to use the tablet?"+
                                " "*5, \_async=True)
            else:
                ...
                tts_service.say("Welcome back! Do you want to use the tablet?"+
                                " "*5, \_async=True)
        ...
```

As you can see, in the first line of the code, we consider the event "Dialog/LastAnswer" taken from the ALDialog APIs, which we will use to get the last answer of the robot according to the user input. Usually, the events are handled by the ALMemory module, in which we link to a callback with the behaviour to be executed when the event is raised, in our case the callback is `handleLastAnswer`. In the *handleLastAnswer* function we can see how with simple string operations we extract the name provided by the user and verify whether it is in the database or not. The employed database has the following structure:

```
    self.patients[name] = {"music":
                                    [(0, "classical"),
                                     (0,"pop"),
                                     (0,"rock"),
                                     (0, "jazz")]
                           }
```

whose main objective is to remember the musical preferences of a given user and make ad-hoc proposals. For instance, when dealing with an already profiled user, if its favourite musical genre was rock, in the current interaction, MARIO will start a specific topic file called `main_rock.top` where it will directly propose to listen to rock songs. The following code shows what we described above.

```
    if name.lower() not in database.patients:
        ...
    else:
        if max(database.patients[name]["music"])[0] > 0:
            favourite_music_genre = max(database.patients[name]["music"])[1]

            ALDialog.unsubscribe('pepper_assistant')
            ALDialog.deactivateTopic(topic_name)
            ALDialog.unloadTopic(topic_name)

            if favourite_music_genre == "rock":
                topic_path = project_path + "/topicFiles/main_rock.top"
            elif favourite_music_genre == "pop":
                topic_path = project_path + "/topicFiles/main_pop.top"
            elif favourite_music_genre == "classical":
                topic_path = project_path + "/topicFiles/main_classical.top"
            elif favourite_music_genre == "jazz":
                topic_path = project_path + "/topicFiles/main_jazz.top"

            # Loading the topic given by the user (absolute path is required)
            topf_path = topic_path.decode('utf-8')
            topic_name = ALDialog.loadTopic(topf_path.encode('utf-8'))

            # Activate loaded topic
            ALDialog.activateTopic(topic_name)

```

```
            # Start dialog
            ALDialog.subscribe('pepper_music_profiled')
        ...
```

From the code above we compute the favourite musical genre of the known user, and based on that we will disable the main topic and activate the corresponding topic given by the favourite genre, allowing the robot to advise that person.

## 4.3 Tablet

Once MARIO has recognized whether it has already met that person or not, it asks "Do you want to use the tablet?". If the answer is positive, then the tablet – represented by the `index.html` file previously opened – will start, otherwise it asks "Do you want to do something else?". For the latter case we will see further details in Section 4.4. More precisely, to start the interaction with the tablet, we use the event "Dialog/LastAnswer", as it happened previously with the name of the user. Through the last response of the robot – *"Let's start with the tablet"* – we launch the script associated with the start of the tablet. The piece of code about what just described is provided below:

```
    def handleLastAnswer(lastAnswer):
        ...
        if "start" in lastAnswer:
            lauch_application(tablet) #lauch tablet application
        ...

    def lauch_application(app):
        with cd(os.path.join(app, scripts)):
            os.system("python demo.py")
        return
```

As we can see, similarly to what we have done in Section 4.2, by means of the `handleLastAnswer` callback, we catch the "start" token in the last answer of the robot, which enable to begin the tablet interaction. This is achieved by invoking the `launch_application` function. Then, when the tablet is running, we will see the following home page:

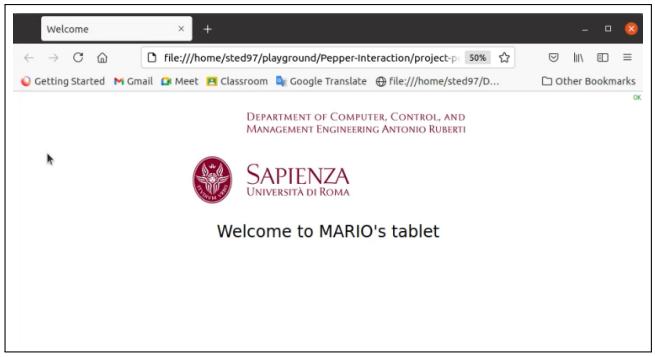



Figure 6: Home page

After few seconds, this page is automatically replaced by the another page asking for your language:

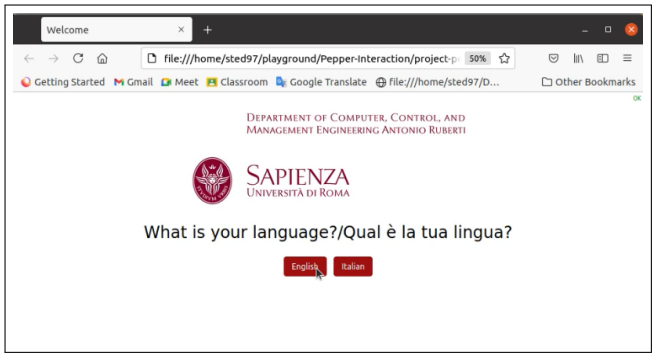



Figure 7: Choose the language

What is shown on the HTML page is handled by the MODIM module, in fact, all the "pages" that we will see from now on are "actions" that we manage through the `ask` and `execute` functions, depending on whether we expect an answer from the user or not. A snippet of code related to the page depicted in Fig. 7 is the following:

```
1    def i1():
2        ...
3        lan = im.ask('language')
4        if lan == "it":
5            im.setProfile(['*','*','it','*'])
6        ...
```

In this case we can see that the action is "language" which will be performed with the ask function because we will set the profile of it according to the choice of the user's language. Now, the entire interaction will follow in the selected language. At this time we arrive at the core part of the tablet where three possibilities are displayed: *reading the news about different topics*, *knowing the forecast of the day* and *playing a game*. What we just described is shown by the following figure.

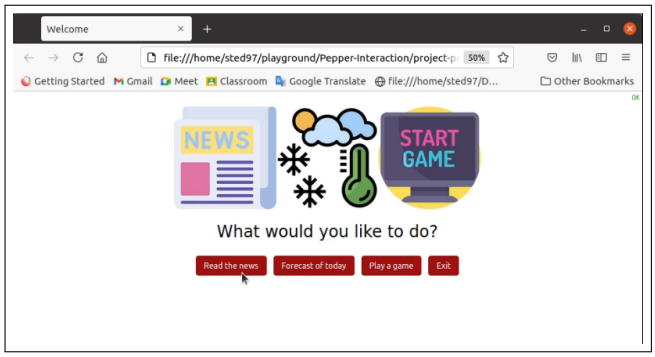



Figure 8: Choose the activity

At any time the user can interrupt the tablet interaction by clicking the exit button. Clicking instead on the "Read the news" button, another page containing the news topics will be shown. The topics are: politics, sport, health, science and technologies, and entertainment.

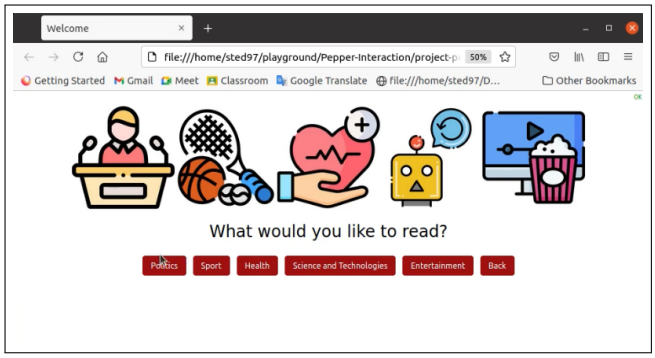



Figure 9: Choose the news topic

For instance, by clicking on the politics category, the output will be something like:

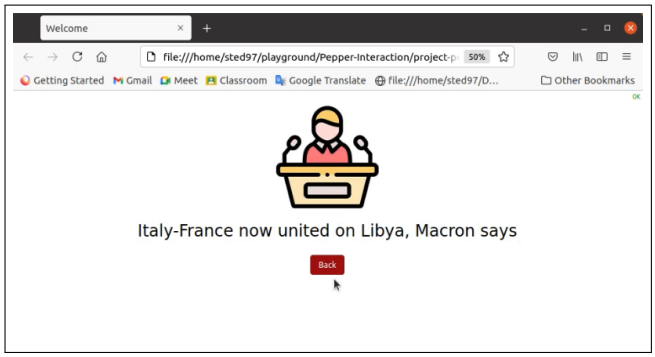



Figure 10: An example of political news.

All the other categories works in an analogous way. At this point, by clicking twice the back button, the user will return to main menu, where it is possible to know the forecast of the day. By performing exactly this actions, the result will be the next one:

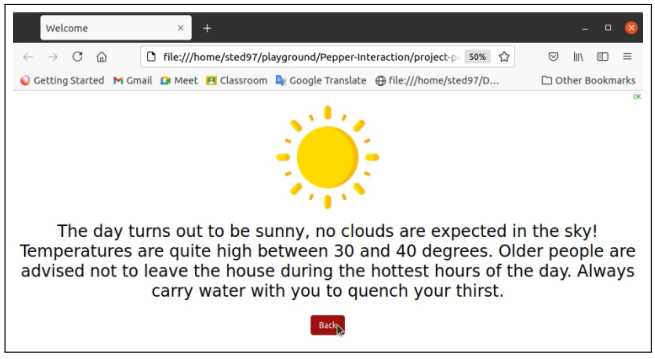



Figure 11: An example of forecast of the day.

Many other outcomes are possible such as cloudy, windy and rainy. Pressing again the back button, the last available activity for the user is to play a game to train its brain. Such game consists in 6 questions: 2 questions in which you have to guess the right word in a sentence, 2 mathematical problems, and 2 image-word association problems.

The starting page of the game, containing the described instructions, is shown below:

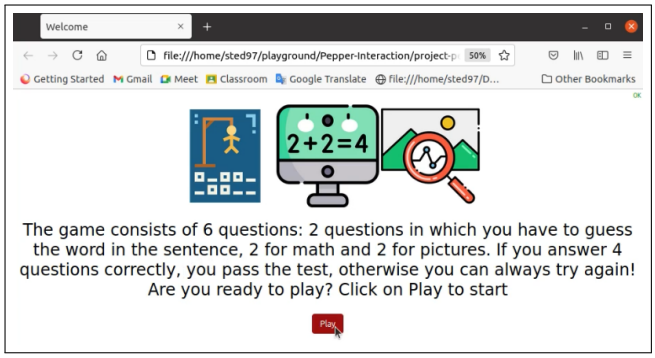



Figure 12: Instructions of the game

Then, by clicking the “Play” button the user will receive two questions of the ”complete the sentence” type, which have the following form:


Figure 13: An example of question in which the user have to guess the right word in a sentence.

Next, after answering to the first two questions, the system will show the third and fourth ones belonging to the mathematical type:

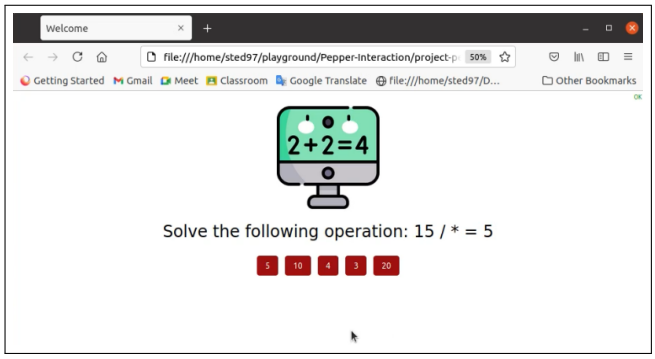



Figure 14: An example of question in which the user have to solve the mathematical operation.

Finally, the last two questions are made as follows:

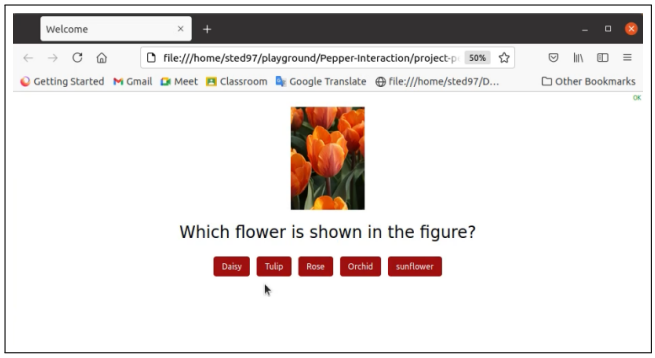



Figure 15: An example of question in which the user have to guess the right word corresponding to the showed image.

During the whole duration of the game, the system keeps track of the user answers and, at the end, it compares them with the corresponding ground truth. If the user correctly answered to at least 4 questions, then a "Congratulations" page is shown, otherwise the system invites the user to keep trying to improve its abilities. In the image below both cases are reported:

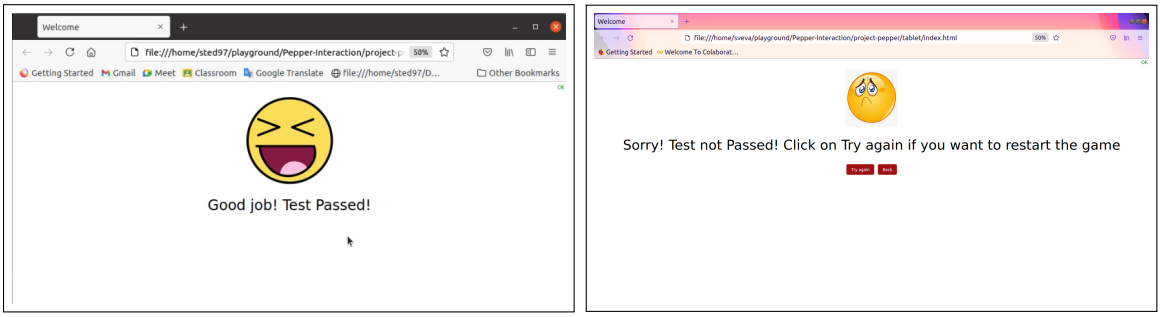



Figure 16: Game results

Now we show some code in which are grouped the various steps of the tablet interaction that we have described above.

```
    def i1():
        ...
        q = 'choose-activity'
        a = im.ask(q)
        while a != "exit":
            if a == "meteo":
                while a != "back":
                    q = "choose-" + str(random.choice(["sunny", "cloudy", "snow",
                                                       "stormy", "foggy", "rainy",
                                                       "windy"]))
                    a = im.ask(q)
                a = im.ask("choose-activity")

            elif a == "news":
                while a != "back":
                    a = im.ask("choose-topic")
                    if a == "politics":
                        im.ask("choose-politics")
                    elif a == "sport":
                        im.ask("choose-sport")
                    elif a == "health":
                        im.ask("choose-health")
                    elif a == "science":
                        im.ask("choose-science")
                    elif a == "entertainment":
                        im.ask("choose-entertainment")
```

```
25
26              a = im.ask("choose-activity")
27
28          elif a == "game":
29              try_again = True
30              while try_again:
31                  a = im.ask("game-intro")
32                  if a == "play":
33                      points = 0
34                      a = im.ask("2-phrases-part1", timeout= 999)
35                      if a == "cat":
36                          points+=1
37                      a = im.ask("2-phrases-part2",  timeout= 999)
38                      if a == "antarctica":
39                          points+=1
40                      a = im.ask("2-maths-part1",  timeout= 999)
41                      if a == "sette":
42                          points+=1
43                      a = im.ask("2-maths-part2",  timeout= 999)
44                      if a == "tre":
45                          points+=1
46                      a = im.ask("2-images-part1",  timeout= 999)
47                      if a == "tulip":
48                          points+=1
49                      a = im.ask("2-images-part2",  timeout= 999)
50                      if a == "amatriciana":
51                          points+=1
52
53                      if points>=4:
54                          im.execute("positive")
55                          try_again = False
56                      else:
57                          a = im.ask("negative", timeout=999)
58                          if a == "try":
59                              try_again = True
60
61                          elif a == "back":
62                              try_again = False
63
64              a = im.ask("choose-activity")
65      ...
```

As we can see, we mainly use the `ask` function, but for the two pages "test passed" and "test not passed" where we use the execute function, because in those cases we do not want any output from the execution of the action. As mentioned above, until the robot clicks on the exit button, the tablet interaction continues. In addition, we also see that in row 33 we keep track of the points that the user makes during the game through a variable "points", so as to show the positive or negative screen at the end. The game can be repeated as many times as the user wants.

Finally, when the user finished to interact with MARIO using the tablet modality, it can always return to the textual or vocal interaction, where the robot has other activities to propose. To notify MARIO that the tablet modality is over, the user should write "finish" in the Dialog view.

## 4.4 Music

As an alternative to the tablet or if the user no longer wants to use the tablet, he can directly interact with MARIO, for example, by listening to music. A code fragment of what just said described above is shown:

```
1   proposal: %question Ok, Let's start with my tablet. Tell me "finish" when you
                        finished with the tablet
2     u1: (finish) Ok, Do you want to do something else?
3       u2: (yes) Do you want to listen some music?
4         u3: (yes) ^goto(music1)
5         u3: (no) Ok, bye bye.
6
7   proposal: %question2 Ok, Do you want to do something else?
8     u1: (yes) Do you want to listen some music?
```

```
      u2: (yes) ^goto(music1)
      u2: (no) Ok, bye bye.
    u1: (no) Ok, bye bye.

proposal: %music1 Which kind of music do you want to listen? Classical, Pop, Rock
                  or Jazz?
  u1: (["classical" "pop" "rock" "jazz"]) Ok, I play it. Tell me "stop" when
                                          you want to stop this song.
    u2: (stop) Ok, Do you want listen something else?
      u3: (yes) ^goto(music2)
      u3: (no) Ok, bye bye.

u: (no) ^goto(question2)
...
```

As you can see from the above code, when the user finishes to use the tablet, or if he no longer wants to use the tablet, the robot will ask "Do you want to do something else?", and if the user responds affirmatively, then MARIO will propose to listen some music. The user can choose from 4 different musical genres: Classical, Rock, Pop and Jazz music. Once the category have been chosen, MARIO will randomly select one of the songs in that specific category and play it. In the meantime, MARIO will dance to the beat of the music trying to catch the interlocutor attention and to make him entertained, moreover, it will also activate a computer vision module to understand whether the person is happy or not while listening to that song. This last part will be described in details in Section 4.5. We report the code with the procedure.

```
    def handleLastInput(lastInput):
        ...
        audio_player_service = session.service("ALAudioPlayer")
        selected_index = str(random.choice([1,2,3]))

        if "classical" in lastInput.lower():
            audio_player_service.playFile(project_path +"/tablet/sounds/classical/
                                          classical"+selected_index+".wav",
                                          _async=True)

            gesture.typeImage = getTypeImage(0)

            if max(database.patients[name]["music"])[0] > 0:
                gesture.favourite = max(database.patients[name]["music"])[1]

            first_elem = database.patients[name]["music"][0][0]+1
            second_elem = database.patients[name]["music"][0][1]
            database.patients[name]["music"][0] = (first_elem, second_elem)

            gesture.doGesture = True
            gesture.doClassical()

        elif "pop" in lastInput.lower():
            ...
            gesture.doPop()

        elif "rock" in lastInput.lower():
            ...
            gesture.doRock()

        elif "jazz" in lastInput.lower():
            ...
            gesture.doJazz()

        elif "stop" in lastInput.lower():
            for i in range(1000):
                audio_player_service.stop(i)
            gesture.doGesture = False
            index += 1
        ...
```

The management of the music is made according to the input of the user. We consider the event "Dialog/LastInput" of ALDialog and, as in the case of LastAnswer , we have a callback function – `handleLastInput` – in which we differentiate the possible cases (Jazz, Rock, Pop, Classical and stop,

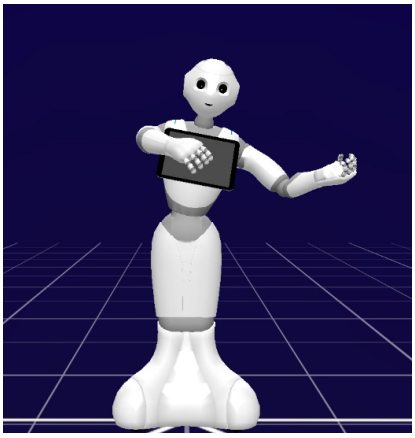
in case the user wants to stop the music). The music starts when we call the function `playFile` of the ALAudioPlayer NAOqi module, and each time the user chooses a song, the database increases the corresponding value associated with that music genre. As previously mentioned, for every musical genre the robot performs a dance, and this is achieved by implementing specific gestures.

Gestures are handled through the joint angles of the robot's body parts, more precisely the Pepper Roll, Pitch and Yaw angles are taken into account. As an example, "RShoulderPitch" indicates the Pitch angle of its right shoulder. We built a total of 6 gestures: 4 for the music, one for the welcome greeting and goodbye and, finally, the one in which Pepper searches for people to interact. The following code shows some examples of our gestures.

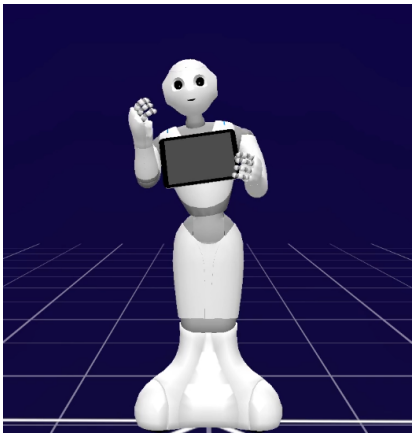
```
    def doHello(self):
        jointNames = ["RShoulderPitch", "RShoulderRoll", "RElbowRoll", "RWristYaw",
                      "RHand", "HipRoll", "HeadPitch"]
        angles = [-0.141, -0.46, 0.892, -0.8, 0.98, -0.07, -0.07]
        times  = [2.0, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0]
        isAbsolute = True
        self.ALMotion.angleInterpolation(jointNames, angles, times, isAbsolute)

        for i in range(2):
            jointNames = ["RElbowYaw", "HipRoll", "HeadPitch"]
            angles = [1.7, -0.07, -0.07]
            times  = [0.8, 0.8, 0.8]
            isAbsolute = True
            self.ALMotion.angleInterpolation(jointNames, angles, times, isAbsolute)

            jointNames = ["RElbowYaw", "HipRoll", "HeadPitch"]
            angles = [1.3, -0.07, -0.07]
            times  = [0.8, 0.8, 0.8]
            isAbsolute = True
            self.ALMotion.angleInterpolation(jointNames, angles, times, isAbsolute)

    def gestureSearching(self):
        for i in range(1):
            jointNames = ["HeadYaw", "HeadPitch"]
            angles = [0.5, -0.07]
            times  = [2.0, 2.0]
            isAbsolute = True
            self.ALMotion.angleInterpolation(jointNames, angles, times, isAbsolute)

            jointNames = ["HeadYaw", "HeadPitch"]
            angles = [-0.5, -0.07]
            times  = [2.0, 2.0]
            isAbsolute = True
            self.ALMotion.angleInterpolation(jointNames, angles, times, isAbsolute)

        return

    def doRock(self):
        jointNames = ["LShoulderPitch", "LShoulderRoll", "LElbowYaw","LElbowRoll",
                      "LWristYaw", "LHand", "RShoulderPitch", "RShoulderRoll",
                      "RElbowYaw","RElbowRoll", "RWristYaw", "RHand",
                      "HipRoll", "HipPitch", "HeadPitch"]

        angles = [1.10, 0.52, -1.36, -1.48,
                  -1.41, 0.27, 1.29, -0.07,
                  0.68, 0.70, -0.26, 0.31,
                  -0.15, -0.17, 0.28]

        times = [1.0, 1.0, 1.0, 1.0,
                 1.0, 1.0, 1.0, 1.0,
                 1.0, 1.0, 1.0, 1.0,
                 1.0, 1.0, 1.0]

        isAbsolute = True
        self.ALMotion.angleInterpolation(jointNames, angles, times, isAbsolute)

        loops = 0

        while self.doGesture:
```

```
59
60          jointNames = ["LShoulderPitch", "LShoulderRoll","LElbowYaw","LElbowRoll",
                          "LWristYaw", "LHand", "RShoulderPitch", "RShoulderRoll",
61                         "RElbowYaw","RElbowRoll", "RWristYaw", "RHand",
62                         "HipRoll", "HipPitch", "HeadPitch"]
63
64          angles = [1.10, 0.52, -1.36, -1.48,
65                  -1.41, 0.27, 1.29, -0.07,
66                  0.68, 1.48, -0.26, 0.60,
67                  0.15, -0.17, -0.43]
68
69          times = [0.5, 0.5,0.5, 0.5,
70                  0.5, 0.5, 0.5, 0.5,
71                  0.5, 0.5, 0.5, 0.5,
72                  0.5, 0.5, 0.5]
73
74          isAbsolute = True
75          self.ALMotion.angleInterpolation(jointNames, angles, times, isAbsolute)
76
77          jointNames = ["LShoulderPitch", "LShoulderRoll","LElbowYaw","LElbowRoll",
78                         "LWristYaw", "LHand", "RShoulderPitch", "RShoulderRoll",
79                         "RElbowYaw","RElbowRoll", "RWristYaw", "RHand",
80                         "HipRoll", "HipPitch", "HeadPitch"]
81
82          angles = [1.10, 0.52, -1.36, -1.48,
83                  -1.41, 0.27, 1.29, -0.07,
84                  0.68, 0.70, -0.26, 0.70,
85                  -0.15, -0.17, 0.43]
86
87          times = [0.5, 0.5,0.5, 0.5,
88                  0.5, 0.5, 0.5, 0.5,
89                  0.5, 0.5, 0.5, 0.5,
90                  0.5, 0.5, 0.5]
91
92          isAbsolute = True
93          self.ALMotion.angleInterpolation(jointNames, angles, times, isAbsolute)
94
95          if loops == 4:
96              self.messageVision(self.vision, self.tts_service, self.typeImage,
                                   "rock")
97          loops+=1
98  ...
```

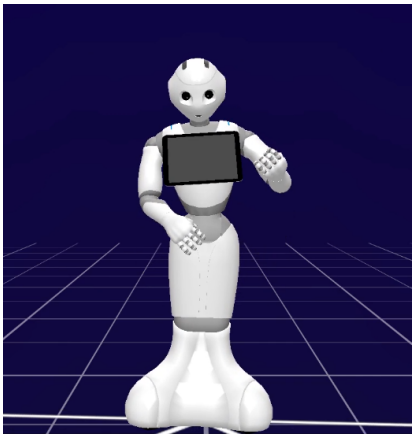
In all the three gestures shown in the code above, we have that angle interpolations needed to connect one configuration with another, hence performing to perform the actual motion, are handled by the ALMotion module. We can also see that the time in which we want to complete the gestures is required. The `doRock` gesture is the only shown musical gesture – the other gestures are very similar – which will be called when the user has chosen to listen to a rock song, and the gesture continues until the song ends or is stopped. Now we show the images related to the 4 musical gestures:



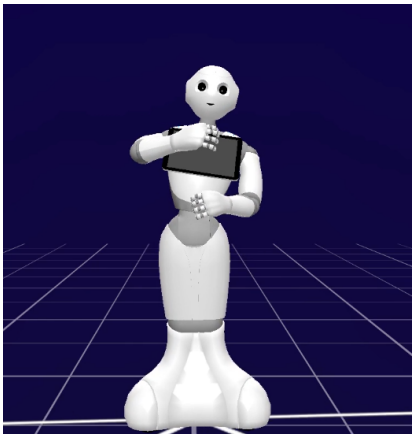
(a) Classical music gesture.



(b) Pop music gesture.



(a) Rock music gesture.



(b) Jazz music gesture.

What we described up to now in this Section refers to a standard conversation with a new user. In the case of an already met (i.e., profiled) user, for which the musical tastes are known, the interaction flow is slightly different to the one shown at the beginning of the Section. In particular, if the profiled user wants to listen to music, MARIO will advise him to listen to his favorite musical genre (the one he has listened the highest number of times, according to the values stored in the database). The following fragment of code shows how this particular interaction is structured:

```
1   proposal: %music1 I remember that you like Rock music. Do you want to play it?
2     u1: (yes) Ok, I play Rock music!! Tell me "stop" when you want to stop this
    song.
3       u2: (stop) Ok, Do you want listen something else?
4         u3: (yes) ^goto(music2)
5         u3: (no) Ok, bye bye.
6       u1: (no) Which kind of music do you want to listen? Classical, Pop, Rock or
    Jazz?
7           u2: (["classical" "pop" "rock" "jazz"]) Ok, I play it. Tell me "stop"
    when you want to stop this song.
8               u3: (stop) Ok, Do you want listen something else?
9                   u4: (yes) ^goto(music2)
10                  u4: (no) Ok, bye bye.
11  ...
```

As previously mentioned, the robot must have activated the topic file corresponding to that user's favorite music, so in this example the robot has activated the file `main_rock.top`, because the user used to like rock music. In fact, we can see at line 1, how MARIO suggests the user to listen rock music. In this case if the user responds "Yes", then rock music will start, otherwise the interaction will be as previously described.

## 4.5 Vision

In our project we built a "fictitious" module with the aim of recognizing the user's emotions during interaction, allowing the robot to understand its preferences. We considered 4 emotions: happy, sad, neutral and surprised. We use the emotion recognition system during the execution of songs in which the robot must understand if the person likes the song or not. The following code shows the class in which we managed our vision module.

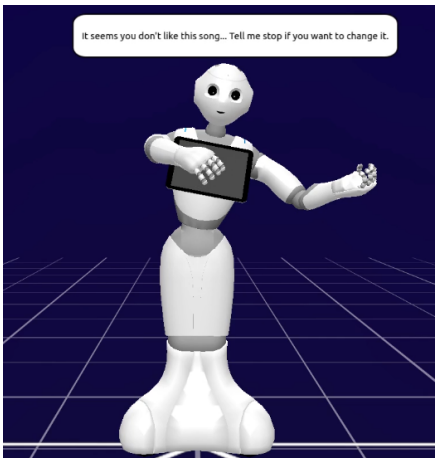
```
1   class Vision:
2       def __init__(self):
3           pass
4
5       def cnnForEmotionRecognition(self, image):
6           classes = ["Neutral", "Happy", "Sad", "Surprise"]
7           prediction = self.forward(image)
8           return classes[prediction]
9
10      def forward(self, image):
11          list_of_candidates = [0, 1, 2, 3]
```

```
12
13          if image == "happyImage":
14              weights = [0.1, 0.5, 0.05, 0.35]
15          elif image == "neutralImage":
16              weights = [0.7, 0.1, 0.1, 0.1]
17          elif image == "sadImage":
18              weights = [0.1, 0.05, 0.8, 0.05]
19          elif image == "surpriseImage":
20              weights = [0.1, 0.35, 0.05, 0.5]
21
22          output = choice(list_of_candidates, 1, p=weights)[0]
23          return output
```

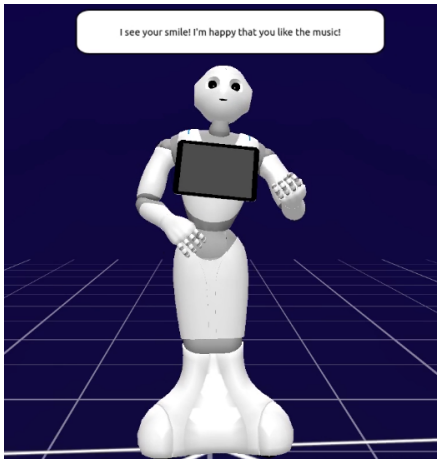
As you can see our idea was to simulate the behavior of a neural network which predicts the emotion of a person. However, we did not have the possibility to interact with the real robot, hence capturing real images of the person to understand his emotion was unfeasible. Therefore, we assumed that if the user is a new user, the network, based on specific weights, will produce a certain prediction. Otherwise, if a person already listened to more than one song, the prediction will be based on the weights adjusted from the previous choices of the user. The following fragment of code shows what we have just described:

```
1   def getTypeImage(index):
2       counter_class = database.patients[name]["music"][index][0]
3       counter_tot = 0
4       for i in range(len(database.patients[name]["music"])):
5           counter_tot += database.patients[name]["music"][i][0]
6
7       weight_happy = counter_class/counter_tot if counter_tot!=0 else 0
8       weights = [weight_happy/2, (1-weight_happy)/2, (1-weight_happy)/2,
                   weight_happy/2])[0]
9       emotions = ["happyImage", "neutralImage", "sadImage", "surpriseImage"]
10
11      if weight_happy != 0:
12          return choice(emotions, 1, p=weights
13      else:
14          return choice(emotions, 1)[0]
15
16  def handleLastInput(lastInput):
17      ...
18      if "classical" in lastInput.lower():
19          ...
20          gesture.typeImage = getTypeImage(0)
21          ...
22      ...
23  # For instance in gesture doClassical we have something like this
24  self.messageVision(self.vision, self.tts_service, self.typeImage, "classical")
25
26  def messageVision(self, vision, tts_service, imageType, current_music):
27      prediction = vision.cnnForEmotionRecognition(imageType)
28
29      if (imageType == "sadImage" or imageType == "neutralImage") and
           current_music == self.favourite:
30          tts_service.say("I see that you no longer like this music that you really
                             liked in the past. Tell me stop if you want to change."
                             +" "*5, _async=True)
31      elif prediction == "Happy" or prediction == "Surprise":
32          tts_service.say("I see your smile! I'm happy that you like the music"+" "
    *5, _async=True)
33      elif prediction == "Neutral" or prediction == "Sad":
34          tts_service.say("It seems you don't like this song... Tell me stop if you
                             want to change it."+" "*5, _async=True)
```

We can see from the code, that as every time the user hears a song, a fake image with the associated "real" class is chosen by the function `getTypeImage`, which will be passed to the neural network for prediction. Based on the prediction of the network, MARIO will then send a message stating whether the user likes that song or not. We show two figures depicting what just described.




(a) Recognize that user does not like this song.




(b) Recognize that user like this song.

In the in which the user is already known instead, and is listening to a song that he previously liked, but now he does not like anymore – for example the prediction of the neural network is 'sad' when the user is listening to its favorite musical genre – then MARIO will ask the person why he no longer likes that music. Accordingly, MARIO asks the user if he wants to stop the song and listen to another one. The figure above shows this last interaction.

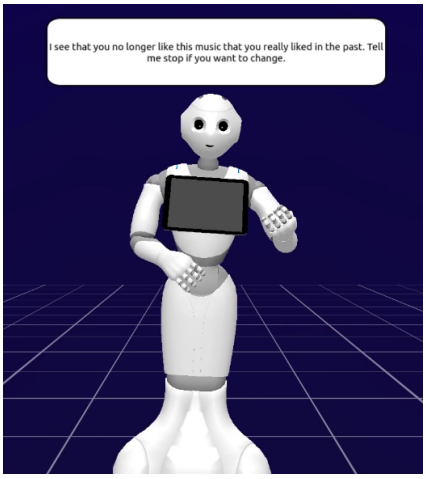



Figure 20: MARIO asks the user to change music since he noticed that he no longer likes.

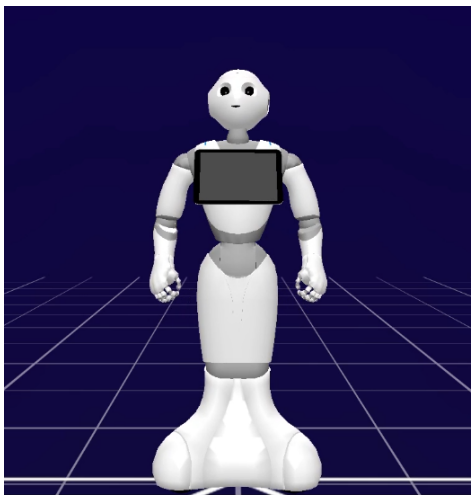
## 4.6 Touch

Finally, we also implemented a module to touch the robot. The robot, on specific body parts, is able to feel the touch from the human, and if this happens MARIO looks around confused. The user can touch the robot on the head, right arm or left arm. The following code shows the module described.

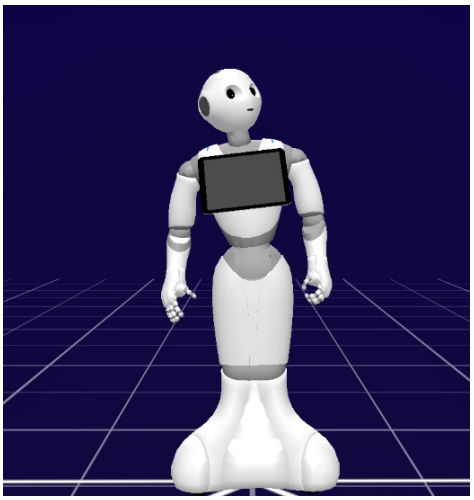
```
1    class Touch:
2        def __init__(self, memory_service):
3            self.memory_service = memory_service
4            self.sensors = {'HeadMiddle': 'Device/SubDeviceList/Head/Touch/Middle/
                                           Sensor/Value',
5                            'LHand': 'Device/SubDeviceList/LHand/Touch/Back/
                                      Sensor/Value' ,
6                            'RHand': 'Device/SubDeviceList/RHand/Touch/Back/
                                      Sensor/Value' }
7
8        def isTouched(self, sensor):
```

```
            try:
                sensor_key = self.sensors[sensor]
                print("Touching %s ..." %sensor)
                self.memory_service.insertData(sensor_key,1.0)
                time.sleep(2)
                self.memory_service.insertData(sensor_key,0.0)
                print("Touching %s ... done" %sensor)
                return True
            except:
                print("ERROR: Sensor %s unknown" %sensor)
                return False
```

Through the `sTouched` function we manage the sensors of the robot that the user will touch, in a "fictitious way", not having a real robot at our disposal. In this case, to handle the touch, we use the NAOqi ALMemory module to enter the value to that corresponding sensor (the touched part of the Pepper body). In addition, below we also show two figures corresponding to Pepper's behaviors when touched.



(a) Pepper when it head is touched.



(b) Pepper when it left arm is touched.

# 5 Results

In this Section we refer to the demo video we realized, publicly available at `https://www.youtube.com/watch?v=2fYorASiQkw`, in which we commented the different phases of the interaction, guiding the viewer to a complete comprehension of the MARIO's capabilities.

# 6 Conclusions

Robotics technology is making huge strides in recent years by providing effective ways to help people in many fields, such as health, education and many others, also allowing them to perform tasks that were too difficult or dangerous. The purpose of the project was to build a Human-Robot interaction that would allow our Pepper robot assistant, called MARIO, to interact with people affected by senile dementia. The software made available by our Professor Luca Iocchi (DIAG, University "La Sapienza", Rome) and by Softbank Robotics, have allowed us to achieve the desired result.

This project has been one of the most practical and interesting that we have done so far, also considering that it is the first time that we directly work on a robot, and allowed us to put forward an architecture which relies on several modules belonging to distinct areas of Artificial Intelligence. Unfortunately, for practical reasons due to the COVID-19 situation, some parts of the project were not implemented in a fully functional way (e.g., the module for emotion recognition). However, the aim of the project was not to implement advanced individual (Computer Vision, Natural Language

Processing or Robotics) modules, already studied in specific courses, but designing an effective system with the focus on multi-modality.

Finally, aware of the difficulties and challenges affecting human-robot interactions – ethics and usability – we have developed our project taking into account all these issues and considering that robots must be included in our society for the purpose of supporting people. We really hope to work on similar projects in the future, possibly by making use of real robots.

# References


[1] Momotaz Begum et al. "Performance of daily activities by older adults with dementia: The role of an assistive robot". In: *2013 IEEE 13th International Conference on Rehabilitation Robotics (ICORR)*. IEEE. 2013, pp. 1–8.

[2] Cynthia L Breazeal et al. "Designing social robots for older adults". In: *Natl. Acad. Eng. Bridge* 49 (2019), pp. 22–31.

[3] John-John Cabibihan et al. "Why robots? A survey on the roles and benefits of social robots in the therapy of children with autism". In: *International journal of social robotics* 5.4 (2013), pp. 593–618.

[4] Kaiming He et al. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[5] Agnieszka Korchut et al. "challenges for service robots—requirements of elderly adults with cognitive impairments". In: *Frontiers in neurology* 8 (2017), p. 228.

[6] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems* 25 (2012), pp. 1097–1105.

[7] Francisco Martín Rico et al. *Robots in therapy for dementia patients*. 2013-01.

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