Pepper: The Chef Assistant

Alessia Carotenuto 1764400
Emanuele Iacobelli 1657799
Veronica Romano 1580844

July 7, 2021

For this project all the students contributed equally.

1 Introduction

The idea behind our Human Robot Interaction project is born after thinking which of the most common daily household tasks could be simplified, speeded up and made more pleasant through the use of a humanoid robot. After careful consideration, we decided to create an assistant in the kitchen, since the culinary field has become very important during these years of pandemic. Our idea was to use a robot capable of following the user to any area of the house, of proposing recipes based on the availability of ingredients in the kitchen and of reading them, so that the user does not have to worry about turning the pages of a recipe book or looking at an application on the smartphone. Of course, we know that there are already many similar applications developed for voice assistants (Alexa first and foremost) but we believe that the use of a humanoid robot can offer a superior experience; just think about the sense of companionship (especially for people forced to stay lonely at home during the pandemic) that is difficult to achieve with simple voice assistants. As stated in the paper [6], a lot of studies have shown that the robot's physical embodiment, its tactile communication and other abilities that embody human-like social signals, can inspire a more engaging and effective interaction compared to an animated character or a voice assistant. The humanoid robot we have chosen to use was "Pepper". This robot has been created and intended for social interactions, i.e. to interact with people by means of different modalities on a day-to-day basis. Compared to other humanoid robots, its aspect is very friendly and could help Pepper to be more accepted by humans. In fact, its aspect has been thought ad hoc, especially regarding the appearance. Not surprisingly, its design has a japanese influence: it has a manga-like big eyes and hip joint that allows Pepper to bow upon meeting someone. Moreover, the shape has been developed to be gender neutral (i.e. it has no explicit gender characteristics) so as to avoid any stereotyping effect. In fact, interesting studies, have been shown that a person tends to consider a robot that looks like the opposite sex more credible, trustworthy and engaging. On the other hand, if a robot exhibits a gender, there is a stereotype-based bias in the expected services the robot should provide. Futhermore, Pepper's voice was designed to be childlike and androgynous, and there are observations that show that people refer to Pepper with different pronouns: his, her and it based on its own perception. Another aspect that makes Pepper suitable to stay in contact with humans is that it has a safety body design with no sharp edges and its CoM is at the base to prevent possible falls. All these features make Pepper suitable for different purposes: hotel receptionist, teacher, bellboy, domestic helper, waiter and so on. For our project, we considered

Pepper as a domestic helper which interfaces with the users through a voice, a tactile and a tablet interaction. We have chosen these interactions because we consider them appropriate for any general user's age, skill and limitations. In particular, the tasks that we developed are:

- Task "Follow me": through a voice command or a touch on the head of the robot, the user will direct Pepper towards the kitchen;
- Task "Let's cook": through voice commands or the use of the integrated tablet, Pepper will give the user the opportunity to choose a recipe based on a particular ingredient (if present at home) or a specific world cuisine (i.e. Italian, Spanish and so on). The user has the possibility to change the ingredient or the world cuisine in the case of a wrong choice. Once the specific recipe has been selected, Pepper will start reading it and will offer, at the end of each sentence, the possibility to:
  - continue the reading;
  - restart the reading of the recipe from the beginning;
  - stop the reading;
  - waiting for the voice command to resume reading;
  - repeat the last sentence.

Finally, Pepper will ask the user to rate the recipe from 1 to 5, so that in the subsequent sessions, the recipes will be listed in order of preference. The detailed version of the functioning of interactions between the robot and the user will be explained in the following sections, together with the software's structure, the implementation and the results obtained in a simulated environment.

2 Related Works

The papers we presented during the course were [2], [4] and [5]. Although we considered them very interesting, they were not functional with our own idea of the project. In this section, we will illustrate the similarities and the differences with some literature works that we found interesting for our project. There are many articles available on the web regarding the interactions between social robots and humans, and many of these take place in the kitchen. A similar idea to ours has been proposed in the paper [1]. The authors of this article have created a robot that gathers information on the users' preferences in the nutritive components of the meals and suggests new recipes with healthy ingredients. The users will use gesture commands, swipe their hand left or right, for interacting with the robot, without the need of touching it, since it is highly likely that the user will have their hands dirty during the preparation of the meal. In our project, we considered the same problem but we used different interactions from the gestures. We give the user the possibility of choosing how to interact with the robot: if the user does not want to touch the tablet during the preparation of the meal (to not dirty it or if he is not able or under condition of using it), he can exploit the vocal interaction, and complete the task in the same way. Of course, there is also the opportunity to complete the same task by using only the tablet. Differently from the authors of [1], even if the tasks are similar, we thought to implement our application on a humanoid service robot which, in our opinion, can have many advantages especially regarding the user's acceptance for the robot. Another interesting paper is [3] in which a vocal assistant which reads the recipes stored in a web service is proposed. In this web service the users can find the recipes, upload their own recipes, create a cooking diary and share a picture of their dishes. The users can directly access the web service by using a PC, a mobile phone or the interactive assistant robot through a vocal interaction. Even in this case, the application allows the users to give commands without using their hands (which makes the food preparation more interesting, engaging and enjoyable) but, as previously mentioned, we think that a humanoid robot like Pepper can produce a better experience. The main difference between our work and this paper is that in our project there aren't other users with whom to interact, while in [3] the structure of the web service is very similar to a social network, that puts in contact all the users and creates a real exchange of information. As we will explain in the *Section 6*, an idea of this kind could be a possible future development for our project. Similarly to this paper, we also used a web service but with different functionalities. In particular, we used a server for exchanging information and commands between the application running on Pepper and the web site running on the Pepper's tablet. In this way, the users do not need to use their hands for tapping on the tablet while cooking. Always in the paper [3], the recipes stored in the web service are written in a format that allows the robot to read them. In detail, the recipe data consists of a table in which the columns represent: the "recipe name", the "image file name", the "description", the "number of people", the "ingredients" and "how to cook". The content of "ingredients" and "how to cook" fields is decomposed in short phrases so as to be more understandable by the users. Instead, in our project, we used a different structure for the recipe data that will be deeply explained in the *Section 4*. The only common thing with this data structure is that we also decompose the procedure in shorter phrases, so that the user can finish preparing each step before passing to the next one. Finally, the last article that we will explain in this section is [7]. The authors, proposed an assistive robot in the kitchen with similar functionalities to the ones implemented in our project but, differently from us, they proposed a system that automatically recognizes the cooking state of the food and thanks to onomatopoeic expressions such as *"kun-kun"* (indicating that the food is almost finished cooking) notify the users of the current state of the meal. In particular, the aim of their project was making the cooking more interesting for children and young adults, but is also used as a cooking rehabilitation system for people with cognitive disabilities. In our opinion the usage of Pepper might be a possible improvement of their work thanks to the fact that it has been created and intended for social interactions.

3 Solution

In this section we will explain the architecture of our project. The main components of our solution are:

- the "Python Client": it is the application that runs on Pepper and contains the main code;
- the "Web Page Client": it represents different web applications that run on the Pepper's tablet at different stages of the interaction with the user during the "Let's cook" task;
- the "Server": it connects the "Python Client" with the "Web Page Client";
- the *"Human_Say"*: it represents the module that allows the simulation of the user's vocal command;
- the *"Touch_Sim"*: it represents the module that allows the simulation of the user's touch on the Pepper's head.

We will now define in detail these components and how they interact with each other during the tasks.

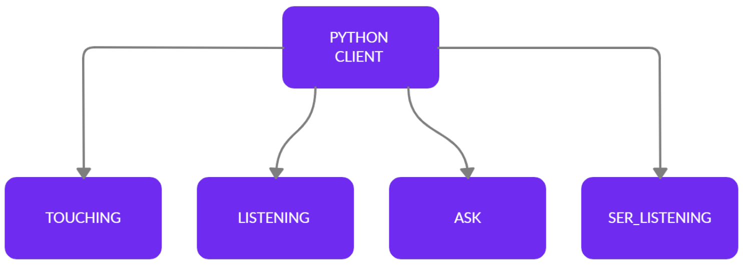


Figure 1: Threads created when starting the "Python Client".

Four threads are created starting the "Python Client":

- *Touching*: it manages the tactile commands coming from the user;
- *Listening*: it manages the voice commands coming from the user;
- *Ask*: it manages the user's answers after a question made by Pepper;
- *Ser_ Listening*: it manages the messages coming from the "Server".

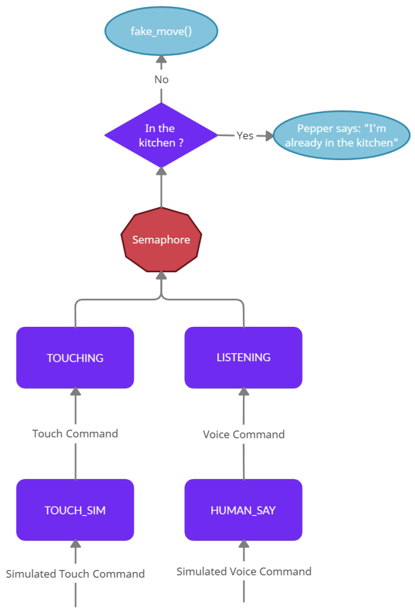


Figure 2: Architecture of the "Follow me" task.

The Figure 2 shows the exchange of information when a voice command or a tactile command is received during the "Follow me" task. The simulated voice command is created by giving as a parameter to the *"Human_Say"* module the string representing the command from the CLI (Command Line Interface). The simulated tactile command is created by starting the *"Touch_Sim"* module from the CLI. When a voice command arrives the thread *"Listening"* captures it by using the function *"wait_ris()"* (explained in the *Section 4*). When a tactile command arrives the thread *"Touching"* captures it. These two threads are controlled by a semaphore that allows (in our specific task) only one thread at time to:

- use the function *"fake_move()"*, if Pepper is not in the kitchen;
- let the Pepper say "I'm already in the kitchen", if Pepper is already in the kitchen.

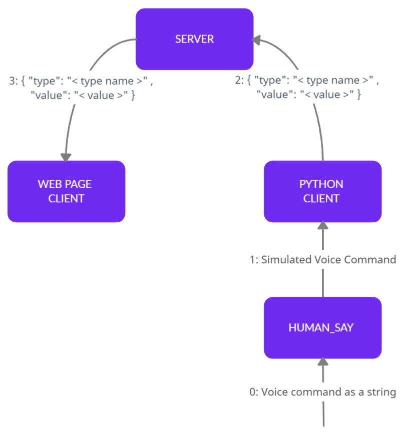

SERVER
3: { "type": "< type name >" , "value": "< value >" }
2: { "type": "< type name >" , "value": "< value >" }
WEB PAGE CLIENT
PYTHON CLIENT
1: Simulated Voice Command
HUMAN_SAY
0: Voice command as a string

Figure 3: Architecture of the "Let's cook" task via voice interaction.

The Figure 3 shows the exchange of information when a voice command is received during the "Let's cook" task. The simulated voice command is created and the fake sentence is acquired by the "Python Client" component thanks to the thread *"Listening"* as explained for the "Follow me" task. The "Python Client" generates a JSON string corresponding to the voice command, sends it to the "Server" and runs the piece of code corresponding to the command. Then, this message is forwarded to the currently active "Web Page Client" that runs the piece of code corresponding to the command.

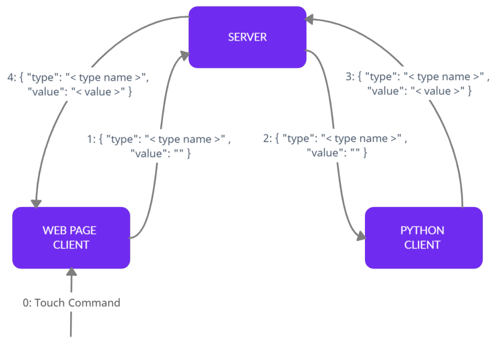

SERVER
4: { "type": "< type name >", "value": "< value >" }
1: { "type": "< type name >" , "value": "" }
2: { "type": "< type name >" , "value": "" }
3: { "type": "< type name >" , "value": "< value >" }
WEB PAGE CLIENT
PYTHON CLIENT
0: Touch Command

Figure 4: Architecture of the "Let's cook" task via tablet interaction.

The Figure 4 shows the exchange of information when a touch command is received during the “Let’s cook” task. A real touch command is given to the “Web Page Client” that only generates a JSON string corresponding to the given command and sends it to the “Server”. Then, this message is forwarded to the ”Python Client” which, thanks to the thread *“Ser_Listening”*, reads the messages coming from the “Server”. After acquiring the message, the “Python Client” runs the piece of code corresponding to the command and generates a new JSON string that is sent back to the “Server”. Finally, this new message is forwarded to the actually active “Web Page Client” that runs the corresponding piece of code.

4 Implementation

Unfortunately, due to the pandemic, we were unable to use the physical robot. Hence, we used a simulated environment to test our code. To see the movements and the phrases said by the robot we used a plug-in of AndroidStudio called “Pepper SDK” that implements an emulator of Pepper. However, by using this emulator, we were unable to show all the Pepper functionalities included in our code such as the turning on and off of the eyes’ leds with different colors and the animations during the phrases. In order to partially solve this problem, we used the OpenCV library to show some of these functionalities and the tactile interaction with the users.

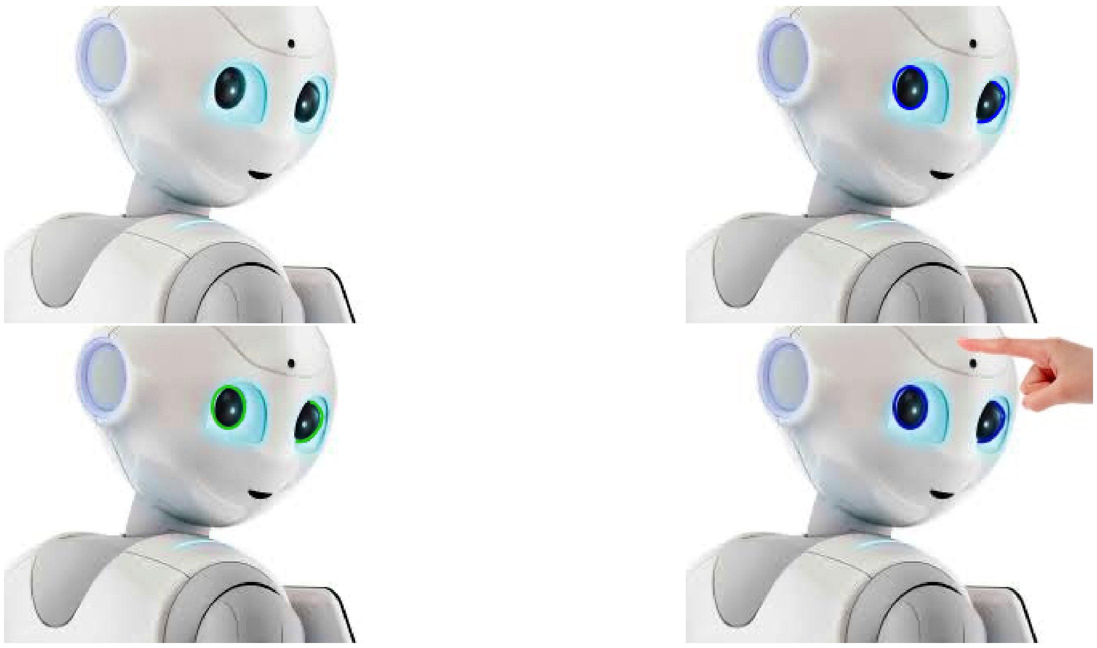


Figure 5: Images used for simulating the use of the leds and the tactile interaction.

In particular, the eye’s leds turn green when Pepper is talking or moving and blue when Pepper is waiting for commands. Regarding the animations, we have implemented the code for the non-animated speech so that we could show the voice interactions in the Pepper SDK emulator, and commented the lines of code for the animated one in order to show the animation we have chosen for each phrase for a real robot. As explained in the *Section 3* for managing the user’s real and simulated commands, the components used are *“Touch_Sim”*, *“Human_Say”*, *“Listening”*, *“Touching”* and “Web Page Client”.

In particular:

- a tactile interaction is managed by:
  - *“Touch_Sim”* that simulates the touch by modifying the value of the Pepper’s variable corresponding to the sensor at the middle of the head;
  - *“Touching”* that captures the interaction by reading the value of the *HeadMiddle* sensor variable through the function *sensorvalue()* in the *pepper_cmd* library;
- a voice interaction is managed by:
  - *“Human_Say”* that simulates the voice command generating a fake event *“FakeRobot/ASRevent”* and setting the values of the *“FakeRobot/ASR”* and *“FakeRobot/ASRtime”* variables representing the Automatic Speech Recognition of the user’s sentence;
  - *“Listening”* that captures the interaction by calling the function *“wait_ris()”*. This function is called after Pepper makes a question and waits for a command. Its operation is simple, it reads a thread-safe LIFO queue, created with the *“Queue”* library, in which are stored the vocal and tablet commands given by the user. Since we have used a LIFO queue only the most recent command is the one executed and all the previous ones stored in the queue are removed. For example if the user, while Pepper is talking, sends a voice command and then taps several times on the tablet, only the last tablet command will be executed after Pepper finishes talking;
- a tablet interaction is managed by:
  - “Web Page Client” that receives a real command from the user and, by using the WebSocket API, sends a JSON message to the “Server” representing the command itself;
  - *“Listening”* that captures the interaction by calling the function *“wait_ris()”* as explained before.

For creating the web applications running on Pepper’s tablet, we used both HTML and Javascript. To allow the communication between the “Python Client” and the “Web Page Client”, we built a WebSocket server in javascript (by using the *Node.js* library) and we implemented a WebSocket client in the “Python Client” (by using the *websocket-client* library available for Python2.7) and in the “Web Page Client” (by using the WebSocket API in javascript). When one of the two clients sends a message to the server, it broadcasts the message to the other client, i.e. the “Web Page Client” sends a message to the “Server” which broadcasts it to the “Python Client” and vice-versa. To exchange information between the clients we used a well known structure that is the JSON format. In particular, the message is a string which represents the commands given by the users in the following form:

- if the senders are the web applications the string is:

{ "type": "< type name >" , "value": "" }”

- otherwise if the sender is the “Python Client” the string is:

{ "type": "< type name >" , "value": "< value >" }

To summarize, all the commands given by the users are processed in the “Python Client” that sends a message to the active “Web Page Client” (through the “Server”) that executes the corresponding code. For example, if a command is given by using the tablet, the “Web Page Client” sends this information to the “Python Client” that sends back a message that allows the web application to execute the corresponding piece of code. Otherwise, if the user gives a voice command, the “Python Client” directly sends to the active “Web Page Client” a message which allows this latter to execute the corresponding piece of code. How the exchange of these messages takes place has been illustrated in the *Section 3*. Regarding storing the information about the recipes, we built two .txt files containing the data structures for representing these information. One file represents the recipe starting from an ingredient and the other the recipe starting from a world cuisine. We tried to create them in such a way it could be very easy to add, modify or remove a recipe. The data structures we have used have the same shape:

< Ingredient Name / World Cuisine Name >$
< Recipe Name > < Evaluation >@
< Recipe Information >/
< Recipe Name > < Evaluation >@
< Recipe Information >%

In detail:

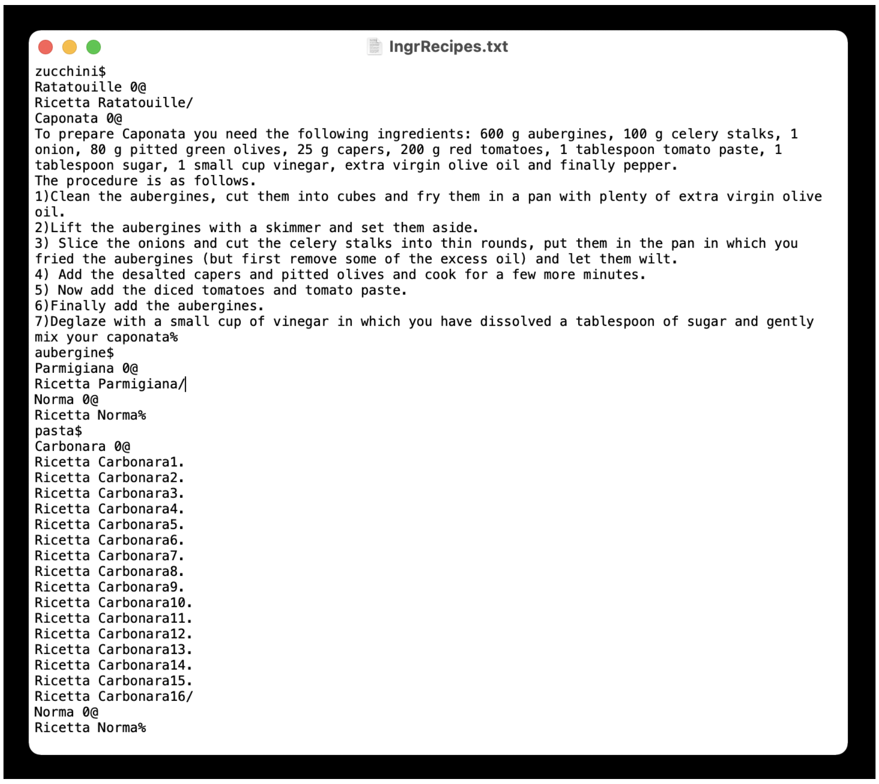
- “$”: represents the end of the row in which the name of a new Ingredient or a World cuisine is defined. All the rows below “$”, until “%”, contain the information of the recipes that refer to this ingredient or world cuisine;
- “@”: represents the end of the row in which the name of the recipe and its evaluation is defined;
- “/”: divides the recipes that refer to the same ingredient or world cuisine;
- “%”: represents the end of the last recipe that refers to the same ingredient or world cuisine.

In particular in the Figure 6, there is the .txt file used for the recipes obtained starting from an ingredient.

From these two .txt files we built, respectively, three dictionaries:

- the first has as key the name of the ingredient/world cuisine and as value the list of the corresponding available recipes;
- the second one has as key the name of the recipe and as value the corresponding evaluation;
- the last one has as key the name of the recipe and as value the corresponding recipe information.

We have used these dictionaries in the *“ChefAssistant.py”*, *“ingredient.html”* and *“worldcuisine.html”* files. In a similar way, to allow the user to better understand the procedure and to give it the time to complete the current step, we divided the recipe information in different phrases, by simply splitting the information at the “.” char. We also want to underline the fact that even if Pepper reads the recipe step by step, the whole recipe is displayed on the tablet, so that the user can also read the full recipe if he doesn't remember some passage. However, some commands for managing the reading of the recipe are available exclusively through a voice command (see the *Section 5* for the full explanation), because in this way the user does not have to use their hands while cooking. Talking about the evaluation of the recipes by the users, both the .txt files have to be modified, in such a way, in the following sessions, the grades will be the most recent ones. In order to do that, we create a function *"writing()"* which takes as input the name of the path of the .txt file, the second dictionary explained before and the user's evaluation. To assign the new grade to the recipe, all the "< Recipe Name > < Evaluation >@" rows of the .txt file are replaced with "< Recipe Name > < New Evaluation >@" by using the *"replace()"* function and the value of the key "< Recipe Name >" in the second dictionary is updated.

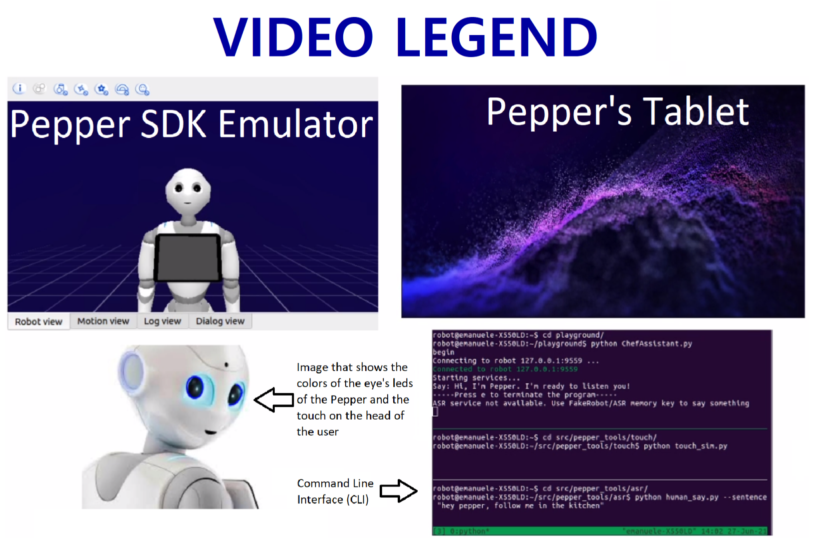


```
zucchini$
Ratatouille 0@
Ricetta Ratatouille/
Caponata 0@
To prepare Caponata you need the following ingredients: 600 g aubergines, 100 g celery stalks, 1
onion, 80 g pitted green olives, 25 g capers, 200 g red tomatoes, 1 tablespoon tomato paste, 1
tablespoon sugar, 1 small cup vinegar, extra virgin olive oil and finally pepper.
The procedure is as follows.
1)Clean the aubergines, cut them into cubes and fry them in a pan with plenty of extra virgin olive
oil.
2)Lift the aubergines with a skimmer and set them aside.
3) Slice the onions and cut the celery stalks into thin rounds, put them in the pan in which you
fried the aubergines (but first remove some of the excess oil) and let them wilt.
4) Add the desalted capers and pitted olives and cook for a few more minutes.
5) Now add the diced tomatoes and tomato paste.
6)Finally add the aubergines.
7)Deglaze with a small cup of vinegar in which you have dissolved a tablespoon of sugar and gently
mix your caponata%
aubergine$
Parmigiana 0@
Ricetta Parmigiana/
Norma 0@
Ricetta Norma%
pasta$
Carbonara 0@
Ricetta Carbonara1.
Ricetta Carbonara2.
Ricetta Carbonara3.
Ricetta Carbonara4.
Ricetta Carbonara5.
Ricetta Carbonara6.
Ricetta Carbonara7.
Ricetta Carbonara8.
Ricetta Carbonara9.
Ricetta Carbonara10.
Ricetta Carbonara11.
Ricetta Carbonara12.
Ricetta Carbonara13.
Ricetta Carbonara14.
Ricetta Carbonara15.
Ricetta Carbonara16/
Norma 0@
Ricetta Norma%
```

Figure 6: The file .txt used for storing the recipes information that refer to the ingredient section.

5 Results

In this section we will illustrate the obtained result considering a standard interaction between a generic user and Pepper. During the explanation of the tasks we will refer to the video (`https://drive.google.com/file/d/1iTN4wMD4gE1OZYWW1l_L55h3GTTUNWjK/view?usp=sharing`) in which we showed all the possible implementations between them. The Figure 7 represents the "Video Legend" showing the emulator of Pepper, the website running on the tablet, the emulator that shows the changing of the Pepper eyes' color and the tap on the Pepper's head and the CLI in which we activate the application and we simulate the user's voice and tactile commands.


VIDEO LEGEND
Pepper SDK Emulator
Robot view
Motion view
Log view
Dialog view
Pepper's Tablet
Image that shows the colors of the eye's leds of the Pepper and the touch on the head of the user
Command Line Interface (CLI)

```
robot@emanuele-X550LD:~$ cd playground/
robot@emanuele-X550LD:~/playground$ python ChefAssistant.py
begin
Connecting to robot 127.0.0.1:9559 ...
Connected to robot 127.0.0.1:9559
Starting services...
Say: Hi, I'm Pepper. I'm ready to listen you!
-----Press e to terminate the program-----
ASR service not available. Use FakeRobot/ASR memory key to say something

robot@emanuele-X550LD:~$ cd src/pepper_tools/touch/
robot@emanuele-X550LD:~/src/pepper_tools/touch$ python touch_sim.py

robot@emanuele-X550LD:~$ cd src/pepper_tools/asr/
robot@emanuele-X550LD:~/src/pepper_tools/asr$ python human_say.py --sentence
 "hey pepper, follow me in the kitchen"
[3] 0:python*                                "emanuele-X550LD" 14:02 27-Jun-21
```

Figure 7: Video legend

We designed this application for a domestic field. We can imagine being in a house wishing to prepare something starting from what is available in our kitchen. After turning on Pepper, it will give us a welcome and advise us that it is ready for receiving a command. To notify the user that Pepper is talking or moving, its eyes' LEDs turn green (min. 00:25), while they turn blue (min. 00:27) when Pepper is waiting for commands. For moving Pepper into the kitchen we can use both the voice command "Hey Pepper, follow me in the kitchen" (min. 01:01) or tap on its head (min. 00:32). In the video, Pepper will move towards a fake location that we can imagine to be the kitchen. After arriving in the kitchen it will say "I'm here" to communicate the end of the task; otherwise if Pepper is already in the kitchen and we give it the same commands as before it will say "I'm already in the kitchen". Then, if we say "Let's cook" (min. 01:27) or we tap the tablet when there is the screensaver (min. 03:56) Pepper will begin the "Let's cook" task and it will ask us if we want to prepare a recipe starting from an ingredient or a world cuisine. After making this choice, through a voice command (min. 04:03) or a tap on the corresponding image on the tablet (min. 01:35), Pepper will ask, respectively, the name of the ingredient that we want to be present in the recipe or the world cuisine from which we want a recipe. We can send this information to Pepper by using a vocal command (min. 01:46) or by writing in the corresponding box on the tablet (min. 04:11). After that, Pepper will propose the corresponding recipes, by saying them out loud and displaying them on the tablet, and it will wait until we make a choice. Even in this case, we have the possibility to select our answer both by using a vocal command (min. 04:21) or by tapping the corresponding image on the tablet (min. 01:53). When we choose the recipe, Pepper will start reading it phrase by phrase while on the tablet we have the whole recipe displayed. At the end of each sentence Pepper will wait for one of the following commands (see the *Appendix "User Guide"* for the specification of each one):

- "Back" (by using a voice command (04:52)/by using a tablet interaction (03:21));
- "Continue" (voice (02:19));
- "Repeat" (voice (02:30));

- “Stop” (voice (03:06)) and “Resume” (voice (03:14));
- "Restart" (voice (02:41));
- “Menu” (voice (05:05)/tablet);
- “Change Ingredient/World Cuisine” (voice/tablet (04:58));
- “Stop cooking” (voice/tablet (05:12)).

Finally, after finishing reading the recipe, Pepper will give us the opportunity to evaluate it. This allows Pepper to list, in order of satisfaction, the recipes in the future sessions. We can evaluate the recipe by using a voice command or by tapping the stars on the tablet (03:49). After the evaluation Pepper concludes the “Let’s cook” task and waits for other commands. To summarize, in following table we list the full interactions shown in the video with the relative time in which they are executed.

User’s voice command/User’s input from the tablet/ Tactile interaction	Video minute
“Tap on the tablet”	03 : 56
“Tap on the head”	00 : 32 00 : 57
“Hey Pepper, follow me in the kitchen”	01 : 01
“Let’s cook”	01 : 27
“Ingredient”	01 : 35
“Zucchini”	01 : 46
“Caponata”	01 : 53 04 : 21
“Continue”	02 : 19 02 : 23 03 : 00 03 : 42 04 : 47
“Repeat”	02 : 30
“Restart”	02 : 41
“Stop”	03 : 06
“Resume”	03 : 14
“Back”	03 : 21 04 : 52
“Ratatouille”	03 : 25
“The user voted 5 stars”	03 : 49
“World cuisine”	04 : 03
“Italian”	04 : 11
“Change world cuisine”	04 : 58
“Menu”	05 : 05
“Stop cooking”	05 : 12

6 Conclusions

In summary, our work is based on the interaction between the user and Pepper in the kitchen. Therefore, we believe that using Pepper as an assistant in the kitchen, can help simplify and speed up a very frequent daily activity. We also considered using Pepper for reducing the sense of loneliness for the people forced to stay alone at home during these years of pandemic. Moreover, we tried to design our application to be used by any kind of user, regardless their age, skill and limitations. Regarding our experience in developing the project, we can say that this work helped us to deepen the concepts learned during the HRI course and also favoured the learning of theoretical notions at a practical level. Unfortunately, due to the current pandemic, it was not possible to test our application on a real robot and this fact, as well as the limitations of the "Pepper SDK" emulator, have caused some disadvantages in carrying out the project. For example we were unable to show Pepper's animations during the speech or to display the changing of the colors of the eyes during the different tasks. This condition forced us to find a way to simulate the lacking interactions but, despite these problems, we can say that we are satisfied with the obtained results. Talking about the possible developments and improvements for our project, we thought of several possibilities. An useful additional interaction could be the gesture recognition: for example, the command to send Pepper to the kitchen can be given with a certain gesture (i.e moving the hand back and forth several times) or to let Pepper continue the reading the "thumb up" gesture can be performed, as well as the "stop" gesture (with one hand) for stopping the reading. Other possible developments could be to display illustrations of the various steps of the recipe on Pepper's tablet during the reading and to change the data structure we used for storing the recipes with a JSON object like "{"type": "Ingredient"/"World Cuisine", "name": "...", "recipe": "...", "vote": "..." } " which simplify the functions for decoding the recipes. Moreover, as shown in [3],a further development, in order to make time in the kitchen more fun, could be to create an application in which multiple users can interact with each other, share recipes, comment on other recipes and so on. Finally, another interesting improvement could be the use of facial recognition to recognize different users and for each of them create an user profile in which store the user's food preferences, habits and possible allergies (similarly to the paper [1]). So that Pepper will be able to recognize the current user and propose the recipes suitable for him.

Appendix: User Guide

This appendix has been designed for any generic user's age, skill and limitations. The following images explain the possible states that can be assumed by Pepper and the possible commands that can be given by the users in each particular state. After each image, we describe in detail the functioning of each command that can be given from each state in the corresponding task.

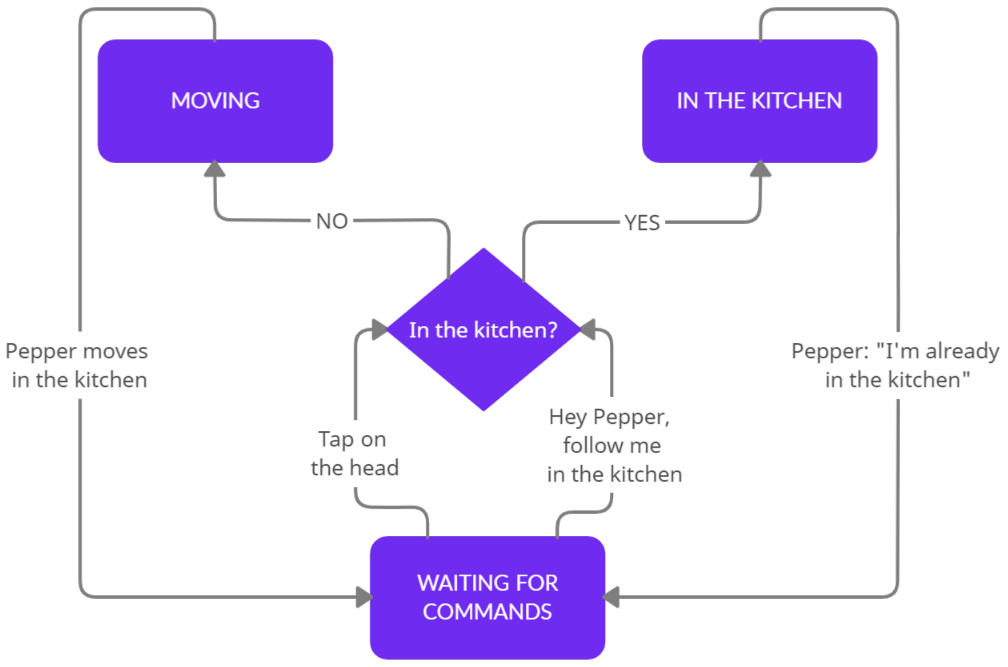

MOVING
IN THE KITCHEN
NO
YES
In the kitchen?
Pepper moves in the kitchen
Pepper: "I'm already in the kitchen"
Tap on the head
Hey Pepper, follow me in the kitchen
WAITING FOR COMMANDS

Figure 8: Task "Follow me": the possible states that can be assumed by Pepper and the possible commands that can be given by the users in each particular state.

For the task "Follow me" (Figure 8) the possible commands are:

- "Hey pepper, follow me in the kitchen" / Tap on the head: Pepper will follow you from its actual position to the kitchen and if it is already in the kitchen notifies the users with the phrase "I'm already in the kitchen".

For the task "Let's cook" (Figure 9) the possible commands are:

- "Let's cook": Pepper understands that you want to start cooking and will ask you whether you want to cook a recipe starting from a single ingredient or from a world cuisine;
- "Ingredient" or "World cuisine": With the first command Pepper understands that the user wants to cook a recipe starting from a single ingredient; with the second one it understands that the user wants to cook a recipe from a world cuisine;
- "< Ingredient Name >" or "< Word Cuisine Name >": Pepper understands the particular Ingredient or World Cuisine that the user has chosen;
- "< Recipe Name >": Pepper understands the particular Recipe that the user has chosen and starts reading it step by step if it is available;

- “Continue”: Pepper reads the next sentence of the recipe;
- “Repeat”: Pepper repeats the last sentence of the recipe;
- “Restart”: Pepper starts reading the recipe from the beginning;
- “Stop” and “Resume”: with the first command the robot will stop reading the recipe and with the second one it will resume reading the recipe;
- “$<$ Evaluation $>$”: after finishing reading the recipe, Pepper asks how much the user liked the recipe, so the next time it will propose recipes starting with the recipe with the highest rating;
- “Back”: Pepper returns to the state in which the user can select the recipes;
- “Menu”: Pepper returns to the state in which the user can select between the ingredient or the world cuisine;
- “Change ingredient” or “Change world cuisine”: with the first command the user can change the chosen ingredient and with the second one the chosen world cuisine;
- “Stop cooking”: Pepper interrupts the task as an assistant in the kitchen.

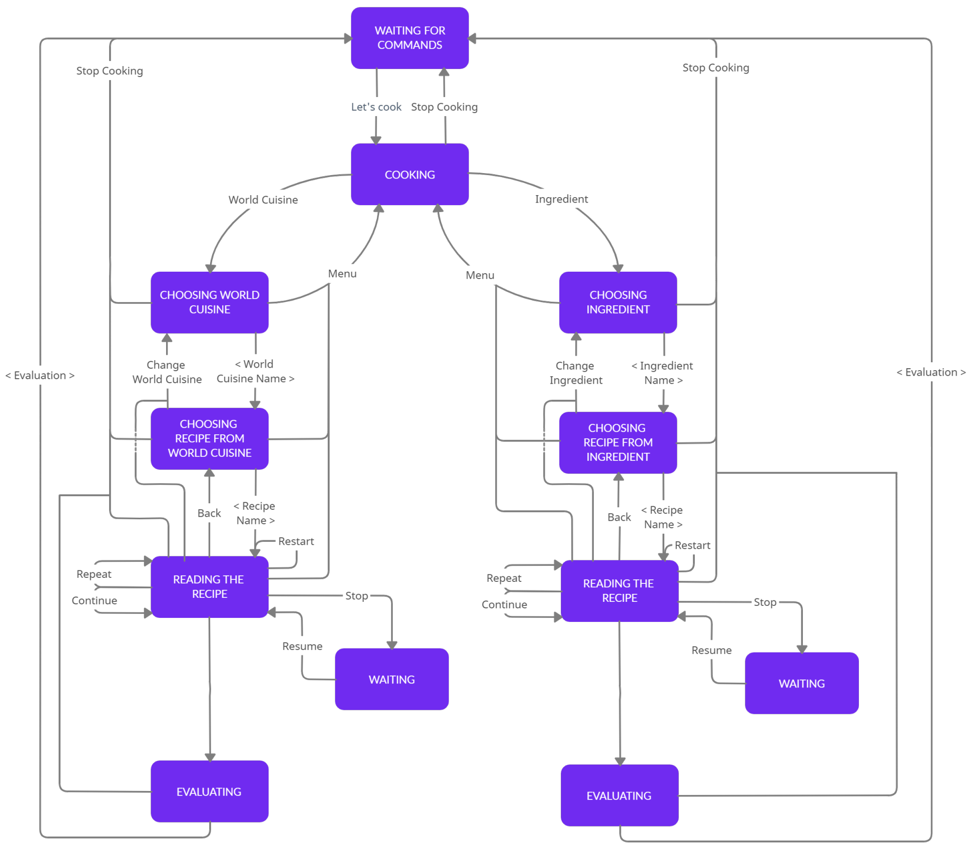


Figure 9: Task "Let’s cook": the possible states that can be assumed by Pepper and the possible commands that can be given by the users in each particular state.

References

[1] Macey Broadwater, Madeline Carlini, and Tate Ewing. Sous-chef: The recipe assistant. In *Companion of the 2021 ACM/IEEE International Conference on Human-Robot Interaction*, pages 622–623, 2021.

[2] Jessica R Cauchard, Kevin Y Zhai, Marco Spadafora, and James A Landay. Emotion encoding in human-drone interaction. In *2016 11th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pages 263–270. IEEE, 2016.

[3] Seiichi Kumagai, Tomoko Sato, Hiroki Sano, Hiroshi Tauchi, Norio Maeda, Yuki Horiguchi, Sachiko Nakagawa, and Masahiko Narita. Cooking assistant service utilizing an interactive robot. In *2017 IEEE/SICE International Symposium on System Integration (SII)*, pages 986–991. IEEE, 2017.

[4] Dylan Moore, Nikolas Martelaro, Wendy Ju, and Hamish Tennent. Making noise intentional: A study of servo sound perception. In *2017 12th ACM/IEEE International Conference on Human-Robot Interaction (HRI*, pages 12–21. IEEE, 2017.

[5] Tayyab Naseer, Jürgen Sturm, and Daniel Cremers. Followme: Person following and gesture recognition with a quadrocopter. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 624–630. IEEE, 2013.

[6] Amit Kumar Pandey, Rodolphe Gelin, and AMPSH Robot. Pepper: The first machine of its kind. *IEEE Robotics & Automation Magazine*, 25(3):40–48, 2018.

[7] Mutsuo Sano Sano, Yuka Kanemoto Kanemoto, Syogo Noda Noda, Kenzaburo Miyawaki Miyawaki, and Nami Fukutome Fukutome. A cooking assistant robot using intuitive onomatopoetic expressions and joint attention. In *Proceedings of the second international conference on Human-agent interaction*, pages 117–120, 2014.