UNIVERSITÀ DEGLI STUDI DI GENOVA

Dipartimento di Informatica, Bioingegneria, Robotica e Ingegneria dei Sistemi (DIBRIS)

Enhancement of Pepper Abilities - Wikipedia

Authors:
Francesca Canale
Alberto Ghiotto
Tommaso Gruppi

Supervisors:
Antonio Sgorbissa
Carmine Recchiuto
Roberto Menicatti

July 13, 2020

Abstract

This report describes our approach to define an application that is able to listen to the user request, to search Wikipedia and to return information from the web page found. Different strategies for presenting information to the user in a natural way and eventually for proposing other related topics have been investigated and implemented by using the Pepper humanoid robot and the Python programming language.

1 Introduction

Verbal interaction is one of the most information-rich modalities for communication and is based on the use of sounds and words to express oneself, especially in contrast to the usage of gestures or mannerisms (non-verbal interaction). Verbal communication usually takes the form of a dialogue and its characteristics include sound, which denotes pitch, speed, and tone, words, which demarcate a set of symbols that are mutually intelligible between the speaker and the listener. For what concerns the context of verbal interactions, there are several considerations that must be taken into account: in particular the individual factors which can drive a conversation and influence the behavior and the reactions of the participants. For instance, from the cultural point of view, there could be differences in reacting to some specific topics: interest, indifference or even inadequacy with respect to a certain argument could vary dependently on participants own culture. Another influencing factor is certainly the age of the people involved, which sometimes could also be related to the amount of usage of a certain type of technology by the individual. For example, it is safe to say that, since middle-aged and elder people interact less with new technologies, they quite often have a less comprehensive knowledge of them.

In the context of the Europe/Japan joint project (CARESSES), coordinated by the University of Genova, algorithms for embedding "cultural competence" in social robots, with a specific focus on verbal interaction, have been developed and tested on a specific robotic platform, the Pepper humanoid robot, developed by Softbank Robotics®. The robot is capable of performing some activities in order to sustain an healthy living for elder people, such as reminding them to take medicines at specific times, encouraging them to live an active life and keeping them in touch with other people. It can also talk about general conversational topics, trying in the meanwhile to learn the user's preferences. In the current implementation, the knowledge of the robot is encoded in the form of an ontology but, in case the user talks about something that is not encoded in the ontology, the robot will reply by stating that it has probably not understood what the user said.
This project aims at dealing with this specific case, by developing an "activity" that searches information online (e.g. on Wikipedia) and reports the information to the user. Wikipedia has a plethora of pages containing information on a vast range of subjects. The Pepper robot can be used to provide such information whenever called for. Thus the main motivation behind this project is for the user to be able to retrieve useful information from Pepper using Wikipedia pages.

Unfortunately, due to the unavailability of the department for safety reasons, it was not possible to continue to develop the application directly on Pepper. Therefore, in agreement with our supervisors, it has been decided to pivot from the initial idea towards a textual interaction implemented as a chatBot, with a behaviour as similar as possible to the original one designed for Pepper. The main differences are that the chatBot will take the user's input from the keyboard instead that from the speech recognition ability of Pepper and will present the output printing it on the screen rather than saying it out loud or displaying it on the tablet. One of the main aspects we have been interested in is the portability of the code within the robot platform.

2 Software Tools

In this project all modules have been implemented by using the Python programming language.

2.1 Pepper & NAOqi SDK

Considering the first phase of the project, different modules have been used inside our functions, namely *helloWorld.py* and *recognizer.py*, in order to interact with Pepper.
To be able to run these modules on a remote PC and to send commands to Pepper, we used the **Python NAOqi SDK** [**1**]. The NAOqi API provides several Python modules to communicate and control Pepper, as well as for applications such as speech recognition, motion, perception and interaction. A short description of what each Pepper service does is presented in the following:

- **ALAudioDevice** : Manages audio inputs and outputs and is used by all other audio modules.
- **ALMemory** : Creates and manipulates memory objects for subsequent use by other modules.
- **ALTextToSpeech** : Facilitates robot speech.
- **ALTabletService** : Displays images and videos on the tablet.
- **Recognizer** : Recognizes user input using Google Speech Recognition.

2.2 APIs

In order to manage both the data contained in the Wikipedia pages and the ones coming from the user, different APIs and external resources were adopted. This allowed us to efficiently manage these data and focus on implementing the different behaviours that the chatBot can perform.
Initially, the suggested API which should allow to search and extract information from any page on Wikipedia using several functions was **wikipedia** [**2**]. Due to some limitations in extracting the sections of the Wikipedia pages, we opted for another python wrapper of the **MediaWiki** [**3**] API which solves the shortcomings of the previous version. The code and description of this library can be found in [**4**].
The APIs provided by the **MeaningCloud** platform [**5**] have been used in order to extract significant contents from what the user is saying and to allow to present the content of the Wikipedia page in a more human-like way with respect to simply reading what is written in it. In particular, we exploited two of the many available APIs, namely "Summarization" and "Topics Extraction", that are used inside the developed functions respectively to extract the main topic from the input received and to present the content to the user in a more suitable way. The first API has been chosen among the available ones in the literature due to its better results in terms of content summarization. In fact, differently from the other APIs tested, it does not present the content by simply discarding some phrases but it also performs a rearrangement of the text, resulting more similar to the human behaviour. The

same considerations apply for the second API, which, unlike many other available ones, in addition to extracting the keyword, performs a categorization of the main words inside the whole input data, allowing us to develop more specific behaviours depending on the category considered.
In the next section, the tools used to face up to these aspects are discussed more in detail.

3 Solution Approach

Figure 1 shows the possible sequences of events and actions that the textual chatBot is able to manage, along with the modules that are used to perform them.
In the expected scenario the user starts the conversation by asking to the application whatever he/she wants to know about. The chatBot starts presenting a brief introduction on the subject requested by the user, if the related Wikipedia page exists, and then performs one of the three different behaviours. In some cases, before performing the behaviour, the application asks to the user if he/she wants to know something else about the topic chosen and if the user's answer is:

- affirmative, it proceeds with the behaviour;
- negative, it proceeds to the final part of the code;
- a question about another topic, it does a research for the related argument restarting the loop. This third case has been implemented only to manage the case in which the user gives an unexpected answer to the question, but initially the conversation was not planned in this way.

If the answer corresponds to the first case, then one of the following behaviours is performed depending on a variable which is toggled at each execution of the main program:

- **presentSection**: this function presents to the user some of the available sections of the Wikipedia page chosen. One possibility is to print all the available sections if requested by the user. Once one section has been chosen, the chatBot briefly presents the summary of its content to the user, exploiting the API described before, and then asks if he/she is interested in another section or not. Ideally the user can ask for all the sections or also for the same section more times in an infinite loop, otherwise if the information provided are enough and the user does not want other information the application proceeds to the last part of the code.
- **presentSuggestion**: this function lists the topics related to the one initially chosen, presents the desired one to the user and proceeds with the last part of the code. Differently from the previous case this function finishes without any possible loop and is executed only once.

In the third behaviour, the chatBot does not ask the user for more information about the initial topic, instead, the function **topicProposer**, once the topic has been introduced, makes an exclamation or a question related to the current topic to the user. Then, it waits for an answer from the user and proceeds in the final part

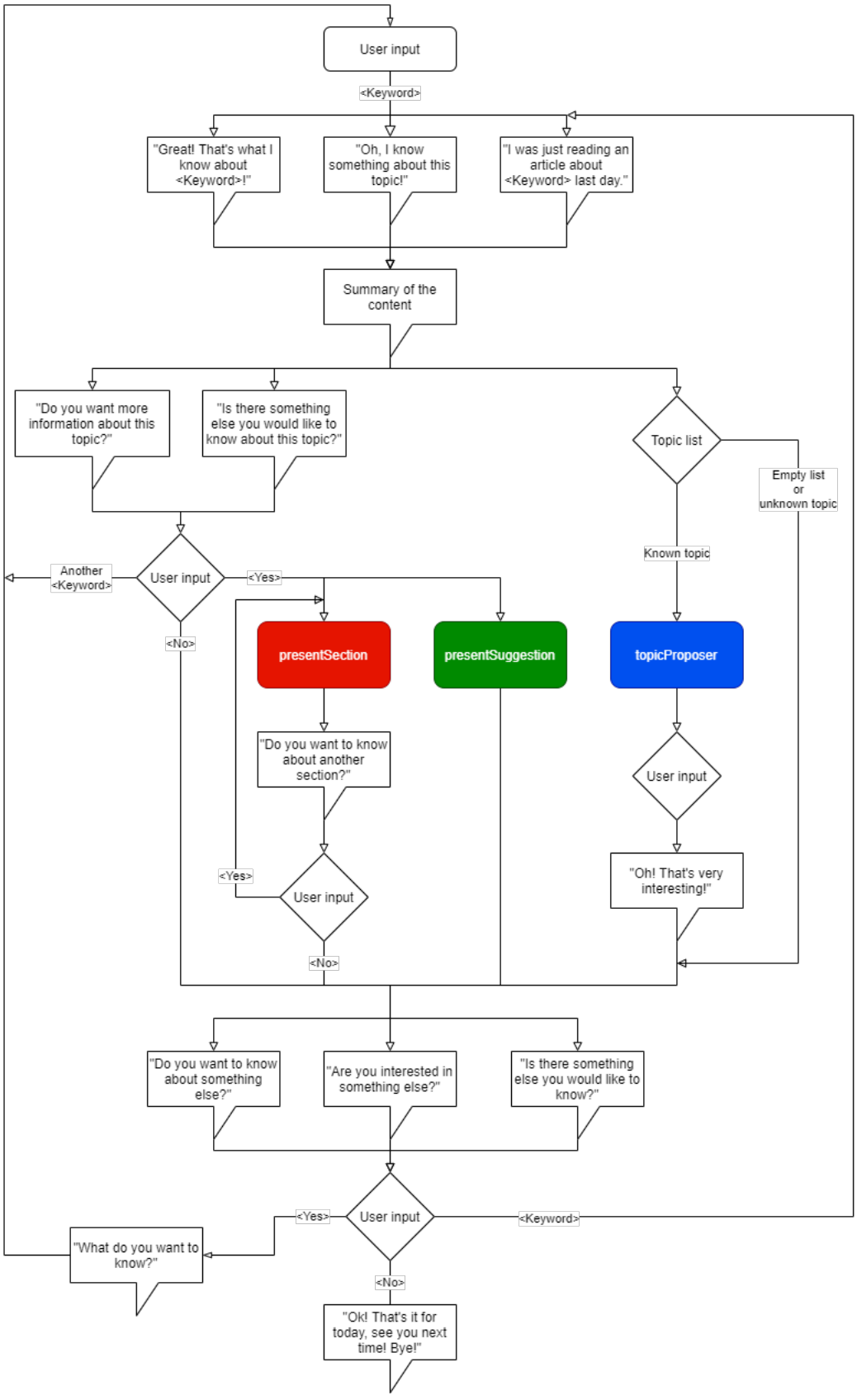

Figure 1: Workflow

of the conversation. If no topic is available in the related Wikipedia page or its type is unknown, the chatBot directly proceeds with the next action.
Finally, in the last part of the structure, the chatBot asks to the user if he/she wants to know something else and if the answer is:

- affirmative, it asks to the user to specify the desired conversational topic;
- directly the argument on which he/she is interested on, the search for the related Wikipedia page starts;
- negative, the chatBot says goodbye to the user and the program finishes.

In order to adopt the cooperative principle, the Gricean maxims are taken into account in all parts of the dialogue. For example, to respect the "Maxim of quantity" and to therefore give the right amount of information about each conversational topic, the user is always asked if he/she wants to know something more and the information is conveyed only if the answer is affirmative. Particular care has been given to provide the best number of sentences in a simple way with nothing more than the information requested.
When interacting with the user, the bot is able to perform two different speech acts (directives and assertives), which makes the interaction more human-like. Moreover, due to the nature of the textual implementation, it is difficult to convey effectiveness during the interaction. This would be different when running the program on the Pepper platform, since the latter allows more emotional speech and content-coordinated movement (i.e. head nods, gaze and arm movements and so on) to achieve a good degree of naturalness. Finally, since this project has been thought to be embedded in the existing CARESSES architecture for the cases in which the user asks for something which is not present in the ontology, our idea is to make the dialogue user-initiated using the not recognized user request as input to the algorithm.

3.1 Functions

Regarding the chatBot, we implemented some functions which exploit the API described above:

- **extractTopic()**: this function takes as input a text and returns a bi-dimensional list of the extracted topics and the relative type by exploiting the "Topic Extraction" API.
- **textSummarization()**: this function takes as input a text and returns a summary of two sentences by exploiting the "Summarization" API. The distributed version returns a summary of five sentences, we chose to decrease it in order to have a briefer content to be proposed to the user.
- **keywordExtractor()**: this function takes the user's raw input, extracts and returns a keyword from what the user inserted, along with a flag stating whether the process has succeeded or not.
- **presentSection()**: this function takes as inputs the list of sections, presents them to the user and, accordingly to his/her choice, displays the summarized

content of the chosen section. This function is able to present the content of all the sections the user is interested in.

- **presentSuggestion()**: this function takes as inputs the list of the topics related to the one being discussed, lists them to the user and presents the one chosen on the screen.

- **topicProposer()**: this function takes as inputs the extracted topic and its type, compares the type with a set of known ones and establishes a brief conversation about this topic.

The last three functions constitute the main behaviours that the textual chatBot can perform and have been implemented in order to make the robot non-repetitive and able to present the information available in the Wikipedia page in different ways to the user.

4 Experimental protocol

Naturally, as it is the standard for this kind of application, we conducted some tests to evaluate the usability of our application. In order to obtain valuable information from the results of our test, we set up the experimental protocol as follows.

4.1 Research question

Our purpose is to evaluate if there are some differences in the interaction with the chatBot between people who have a technological background and people who have not. In the analysis we consider the usability of the application as terms of how much users appreciate the interaction with it.

4.2 Hypothesis

The hypothesis we aim to verify is whether people who have technological competencies and experiences will evaluate the usability in a more positive manner since they better know the mechanism and the techniques involved in its implementation. On the other hand, it has to be considered that, given the better understanding of the various nuances involved in the application, the "technological competent" subjects, could have higher expectations and a more critical approach relative to some aspects of the project.

4.3 Experimental Units

We decided to divide the participants in two groups based on their level of confidence towards technology's state of the art. In particular, we discriminated between the two groups on the basis of an existence of a background in ICT engineering studies. The individual to be tested were subjected to a small screening prior to the actual selection to guarantee a minimal set of prerequisites. The first and foremost was about the ability to speak English to a certain level of fluentness, in order to avoid problems due to the misunderstanding of the sentences.
A total of 22 individuals completed the interaction with the chatBot and were equally

distributed between the two groups both consisting of 8 males and 3 females. The ages of the participants of the technological group vary between 21 and 40, while for the other group 7 people are between 21 and 30 years old and 4 are between 51 and 60.

4.4 Variables

The independent variable in this case is represented by the background of the participants while the dependent variable is represented by the performances of the algorithm in terms of usability. The covariates are instead represented by the age, the gender and the occupation of the experimental units.

4.5 Conditions and Study Design

Since our study focused on studying how the technological background of a subject influenced the interaction with the chatBot, the condition is technological background in contrast to non-technological background. In our approach, rather than assigning the condition to the experimental units, it was necessary to divide the subjects in two groups based on the condition. For this reason, since the two groups are separated, every subject will test only its condition. Therefore this methodology falls within the "between subjects" category.

4.6 Measures and Metrics

The interaction with the chatBot lasted 7 minutes and was preceded by a brief explanation of what a chatBot is and how to use it. During the interaction the user was not allowed to ask any question to the moderator. This approach has been chosen since the overall implementation of the algorithm has been thought to be simple for every type of user and consequently we wanted to evaluate also this aspect.
At the end of each test the participants answered to some personal questions about themselves and completed a System Usability Scale (SUS) which consists of a 10 item questionnaire with five Likert scale options from "Strongly disagree" (1) to "Strongly agree" (5).

SUS Questionnaire
(1) I think that I would like to use this system frequently.
(2) I found the system unnecessarily complex.
(3) I thought the system was easy to use
(4) I think that I would need the support of a technical person to be able to use this system.
(5) I found the various functions in this system were well integrated.
(6) I thought there was too much inconsistency in this system.
(7) I would imagine that most people would learn to use this system very quickly.
(8) I found the system very cumbersome to use.
(9) I felt very confident using the system.
(10) I needed to learn a lot of things before I could get going with this system.

5 Results

The System Usability Scale is not diagnostic and it won't tell anything about the specific problems of the application. Instead, it will give an idea of how much work needs to be done to improve the usability of the tested application.

5.1 How to calculate the usability score

The users have ranked each of the 10 templates questions above from 1 to 5, based on their level of agreement.
Here is how to compute the SUS score to asses their experience:

- For each of the odd numbered questions, subtract 1 from the score.
- For each of the even numbered questions, subtract their value from 5.
- Take these new values and add them up to find the total score.
- Multiply the total by 2.5.

5.2 How to evaluate the usability score

The average System Usability Scale score is equal to 68. If the score is lower than 68, then there are probably some serious problems with the application that should be addressed. If the score is above 68, then there's a good base from where to start. Here's an overview of how the SUS scores should be measured:

- 80.3 or higher is optimal $\rightarrow$ people love the application and will recommend it to their friends.
- Around 68 is acceptable $\rightarrow$ the usability is good but could improve.
- 51 or under is a clear signal that something is wrong $\rightarrow$ improving the usability should be a priority.

5.3 Our SUS score

Once the SUS score has been computed according to the previous description we obtained an average score equal to 69.77 for the group without the technological background and equal to 73.18 for the other one. Both these two values correspond to acceptable results according to the above mentioned evaluation method.
In figure 2 the distribution of the scores considering the four evaluation ranges is shown: it can be noted that the majority of them belongs to positive evaluations (scores grater than 60) and that in both groups the acceptable range is the most common one. In any case it has to be considered that 4 units have evaluated the chatBot in a negative way reflecting in some limitations in terms of usability of the proposed algorithm that should be addressed more deeply.

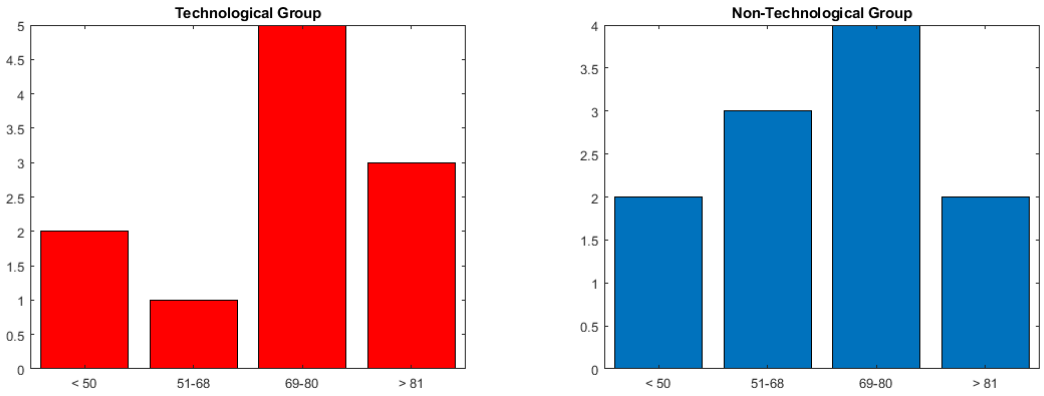

Figure 2: SUS scores distributions

In the images reported below we highlight the answers of the two groups to some interesting questions.

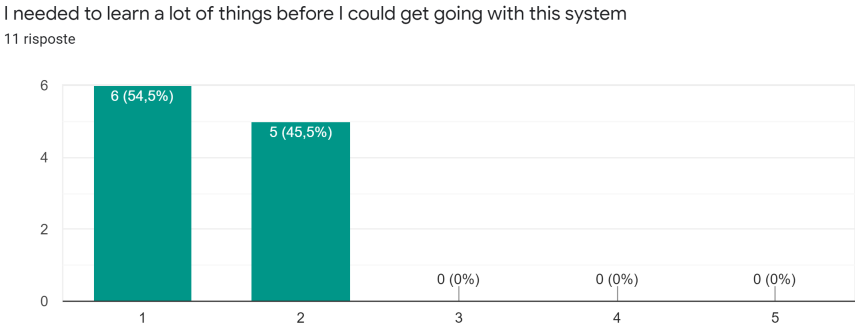

Figure 3: Question 10 - Answers of the technological group

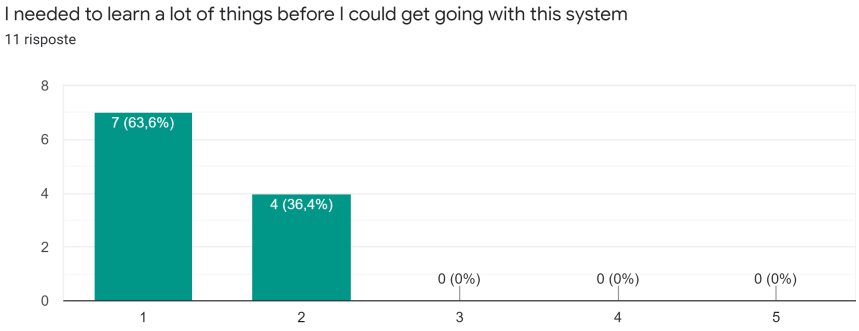

Figure 4: Question 10 - Answers of the non-technological group

From the answers of question 10, presented in Figure 3 and 4, it can be noted that all the units of both groups think that there is no need to learn anything to be able to interact with the chatBot reflecting in some way the simple structure we wanted and demonstrating that the system could be used by everybody regardless their previous technological knowledge.
Nevertheless, the answers of question 3 presented in Figure 5 and 6, tell us that there's still room for improvement when it comes to making the application as easy as possible to use.

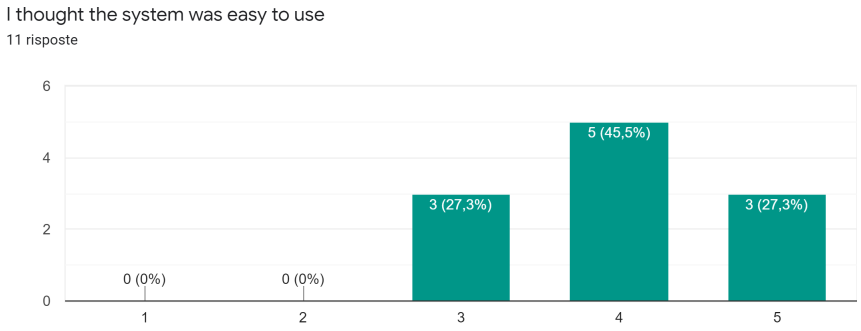

Figure 5: Question 3 - Answers of the technological group

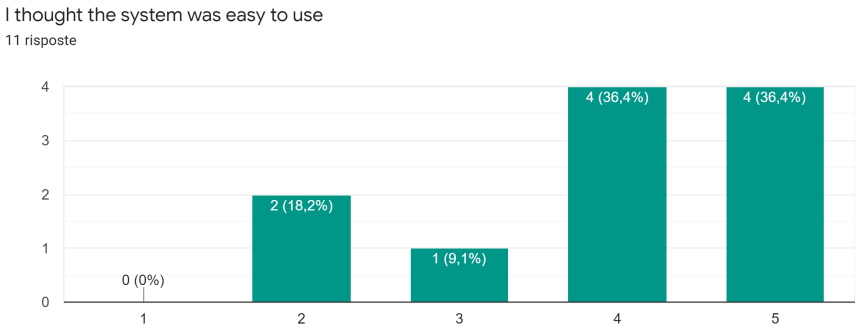

Figure 6: Question 3 - Answers of the non-technological group

5.4 T-Test

In order to correctly evaluate if there are any differences between the two groups of users, we performed an equal variance T-test, since both groups are characterized by the same number of units. The T-value obtained from the computations involved in this test is equal to 0.0071 while the degrees of freedom are equal to 20. Since our T-value is really small we can accept the null hypothesis and consequently we can assert that there are no differences between the two groups considered or that the groups are similar.

5.5 Performance

During the test some parameters have been collected in order to extract some useful information about how the users reacted to the application. In particular we decided to gather data about:

- How many times users asked the chatBot for more information about the chosen topic.
- How many times the users asked the chatBot to know something about another topic.

These two information are outlined in the graphs in Figure 7. From these pie charts it can be noticed that most of the times users are engaged and interested in the conversation and so they usually ask for other information. This could be interpreted as a positive feedback from the point of view of the quality and also of the quantity of the information provided to the users

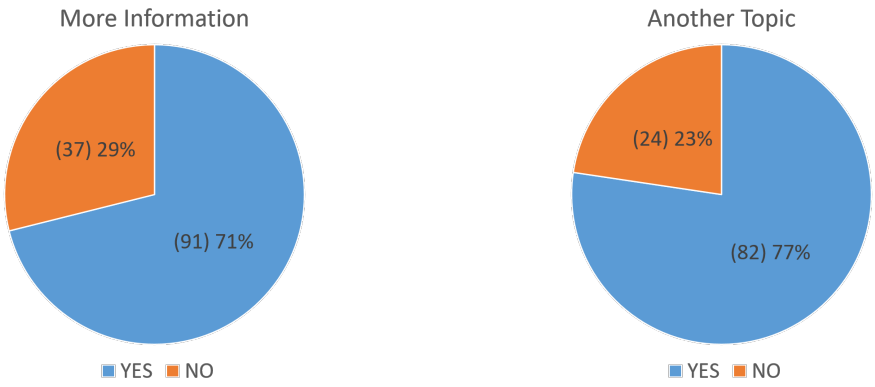

Figure 7: Parameters

6 Future improvements

The first future improvement would be to change the code in order to make it suitable to work with Pepper. This would simply consists in replacing the Python statement *raw_input()* with a call to the related function defined in *recognizer.py*, namely *listen()*, in order to allow the robot to detect what the user is saying. The other aspect that must be taken into account is when the robot, instead of the chatBot, has to interact with the user: in this case it would be sufficient to replace the Python function *print()*, which displays the phrase on the terminal, with a call to the **ALTextToSpeech** service by calling the related function *say()*, which makes Pepper say the content passed as parameter.
Another possible improvement could be to make the robot move while presenting some content or interacting with the user in order to make it even more human-like. Finally, Pepper is equipped with a tablet, which in the first phase of the project had been used to show the images related to the discussed topics. The tablet could also be used to improve the interaction with the user in different ways. For example, it could be employed to display the list of the sections found in a Wikipedia page or to show some related videos. Consequently, a possible future analysis can be done on the hypothesis that the implementation on the humanoid robot will be preferred, with respect to the chatBot, by the majority of the population.

6.1 Critical aspects

Our algorithm is characterized by some traits that might define it as limited, for example, the inability to learn concepts from the conversations or to automatically understand which is the context or the situated language that it should adopt. These attributes are due to the nature of the application, the requirements of this project and the given temporal constraints and are therefore implemented in this way to maintain the interaction as simple as possible. For the same reason, this implementation does not use any planning algorithm or natural language processing that would be useful for better understanding the meaning and the arrangement of the words. Again, in the same way, the dialogue is kept limited, characterized by specific patterns and usage of words to lead the conversation and bound the user to answer in a specific way. Nevertheless, different assertions, questions and behaviours have been implemented in order to make the robot less repetitive and consequently

as human-like as possible.

7 References

References

[1] Python NAOqi SDK, http://doc.aldebaran.com/2-5/dev/python/index.html

[2] wikipedia 1.4.0, Wikipedia API for Python, https://pypi.org/project/wikipedia/

[3] MediaWiki API, https://www.mediawiki.org/wiki/API:Main_page

[4] Wikimedia, https://github.com/barrust/mediawiki

[5] MeaningCloud API list, https://www.meaningcloud.com/developer/apis

[6] MeaningCloud Python SDK, https://github.com/MeaningCloud/meaningcloud-python