Report
Spoken human-robot interaction project

Fabrizio Casadei

Matriculation number: 1952529

SAPIENZA
UNIVERSITÀ DI ROMA

February 23, 2021

Chapter 1

Overview

This report describes the work done for the "Spoken Human-robot interaction" homework, which consists in designing a *Task oriented Spoken dialogue* system.
The system is fitted around a specific scenario, arbitrarily chosen. In this case, we have managed a chatbot which provides several functions for the office.
In details the Agent is able to perform the following tasks:

- Give information about itself, like the own name,
- Tell all the information that it knows about the office (objects, their positions, possible additional data),
- Respond to a request about a specific object, telling if it's present or not, in the case, position and other information,
- Save new objects for the office,
- Delete, objects that are registered,
- Read the agenda of the office worker,
- Add new appointments, storing information about the name of the event and the date,
- Remove already stored events in the agenda,

Here we define the principal operations that are been computed to realize this project. First is performed a *speech recognition*, that converts the audio input to the corresponding string, using a *ASR*(Automatic Speech Recognition).
After having defined a way to achieve the text input, the corresponding frames are built, concerning the chosen scenario.
Frames it's one of the knowledge representation formalism, that is applied in the *NLP*, are used to model situations in a specific context. The exchange of information, can take place by a process called *slot filling*, which means, in order to complete the information about a subject, a certain number of pieces should be given, and such pieces are the slots of the frame.
Finally a *text to speech* module is defined to realize the *Natural language Generation*.

Chapter 2

Implementation

The construction of the project can be divided into 4 main modules, that cooperates to reach the final goal of a *Spoken Dialogue system*. in this chapter, we will see all the relevant implementation aspects related to the developed system.

2.1 Listener

The first component that we're going to describe, it's the one in charge of performing the *Automatic Speech Recognition*. To define this module we've used the library "speech_recognition" that provides different kinds of recognizer, used to convert the audio, taken from a microphone source, to the corresponding text.
The recognizer used is the one developed from *Google*, that achieves a high level of precision in the transcription of user dialogues.
Using this component we are interested in acquiring commands from the user to interact with our office bot, therefore, strings that are parsed from the *SDS Agent*.
The message from the user can be not recognized, in that case, an error message has been provided for informing that the command must be repeated.

The system booting, consists, of an initial message, written in the console and spoken through the speakers of the device, and immediately after, a listening session start, that is interrupted for the parsing, after 1 second of silence from the message.
Every important indication for the user is provided both in text and audio way.

2.2 SDS Agent

After having recognized the user's command, the agent takes over this string to generate the appropriate response, using an approach rule-based driven.
This process is computed through two main steps: the command match, and the corresponding act.

the first step is computed **matching** the command with templates that the agent knows.
These templates are defined as a dictionary, in which the keys are the name of acts that must be performed, and the values are lists of regular expressions, used to associate the command received to a certain act name. One possible example is the following:

```
"ask_about_object":["where is the (?P<object_name>[\w\s]+)",
  "tell me information about (?P<object_name>[\w\s]+)"]
```

Through the regular expression, it's also possible to extract important information used for the slot filling of the second step. In the example above, a command is categorized as "ask_about_object" and the object name, is extracted during the match operation.

In this process two different situations can happen, a command is matched with one of the templates defined, or none is the correct template referred to the particular user input, in that case, a predefined message is used and spoken through the speaker object of the agent, saying: "I didn't understand what you said, please repeat".
After having informed the user that there was no matching for the command, it starts again to listen, for a new command. At each iteration is possible to stop the conversation, using one of the shutdown commands that are been defined, like "bye-bye" or "exit".

After having received positive feedback from the matching part, follows the act step. What is important here, is the key of the template given from the match and information extracted thanks to regular expressions.
for each key, a specific frame is defined, and a slot filling operation is done using the extracted information from the templates. Whether a specific act involves a modification or the extraction of data from the knowledge base, several useful function to manage the database are invoked. Once again, referring to the example above, a possible response from the act function could be:

```
"The following information about Russell & Norvig, AI book are available:
the position is: Library
additional information are: A famous book about the Artificial intelligence,
        used to prepare the AI exam of your master's degree.
```

These concepts are applied for all the possible task performable, introduced in the first chapter.

Moreover, in the system, it's possible to enable, through an internal boolean flag, the print of the dependency tree about every spoken sentence. To compute this has been used the "Spacy" library, a very reliable module used for dependency parsing.

To conclude this section about the agent, below, it's shown a complete example of interaction "user-office robot" involving two agenda commands.

```
Initializing the chatbot...
Hi human, i'm an office robot, how can i help you?
Speak now:
audio input: save this appointment
save_event
Please tell me the event that you want to save in the agenda
Speak now:
audio input: soccer game
is the soccer game the event that you want to save?
Speak now:
audio input: yes it is
yes it is
Please tell me now its date
Speak now:
audio input: 22nd February
is the 22nd february the date?
Speak now:
audio input: yes it is
yes it is
The event has been successfully added in the knowledge base
```

"Save a new event in the agenda"

```
Speak now:
audio input: show me my agenda
show_agenda
The following are you're appointment saved in the agenda:

Appointment number 1
 event: Meeting with university teacher,  date: 20 May

Appointment number 2
 event: oculist visit,  date: 21 march

Appointment number 3
 event: visit to grandma,  date: 5th february

Appointment number 4
 event: soccer game,  date: 22nd february

Speak now:
audio input: ok thank you bye-bye
shutdown
shutdown the chatbot...
```

"Read all the event stored in the agenda"

2.3 The knowledge base

The knowledge base, or KB, it's a main part of the system, that allows the agent of having specific knowledge about a certain scenario. Hence, the KB and the agent, sometimes are seen as a unique entity, in which the kb represent it's own memory and drives the agent choices and responses.

This knowledge base is represented through two local relational database, built with the "sqlite3" library. The first represent information about the office, and it's defined in these terms:

- **ID**, the identifier (integer) of each tuple of the relation "office",
- **Object**, a string field, that describe the noun of the object,
- **Position**, a string field, that explicit the object's position,
- **Information**, a string field, used to store additional data about objects.

The second local database, has been used to store notions about the worker agenda, and it's made up by:

- **ID**, the identifier (integer) of each tuple of the relation "agenda",
- **Event**, a string field, that gives information about a certain event stored in the agenda,
- **Date**, a string field, used to persist the date of a certain event,
- **Information**, a string field, utilized to represent additional information about the agenda,

This permits making persistent data about our scenario, through the interaction with "office robot".
Several useful functions are been provided for both the database, like the insertion of data by the definition of fields, deletion by the event or object name, so on and so forth.

Initially, the knowledge base is represented by empty databases, filled with "save operations", and reduced with the "delete operations" called from the agent.

2.4 Speaker

The final module which is involved in the whole process is the speaker. Different internal messages after being created, have to reach the user, through a text to speech operation. For doing this, a library called "pyttsx3" has been used.
The speaker provides a function to make this conversion, called many times from the agent, for example, to respond a question about what objects has been registered in the office. This operation is fundamental to close the communication cycle between the Agent and the user.

The *TextToSpeech engine* is configured for having specific properties, that concern, the audio volume, the rate of the speech and the type of the voice that we want to use.
Finally has been implemented the possibility of put in a queue messages for the users, to use in another moment.

Chapter 3

Conclusion

In the end, we can be satisfied with the result obtained, using a very simple structure of the system which achieves a coherent vocal interaction with the user, exchanging messages to execute and perform tasks.
Beyond the work done, it's easy to understand that a similar approach can't be suitable for more ambitious projects, in which a methodology, strictly based on the parsing of the user message, through *Language Understanding* and *Dialogue management* that analyzes the morphology, the syntax and the semantics, it's well suited.
It's also possible to engage probabilistic models, and machine learning systems as well, in order to train classifiers able to detect correct classes for the user requests.

All in all this project, even if in a simplistic manner, shows the main steps needed to realize *Task oriented Spoken dialogue* systems, that are increasingly populate the daily routine of any person, leading to a greater interest in their development and improvement.