

Spoken Human-Robot Interaction project Report

OLGA SOROKOLETOVA
1937430
Sapienza Università di Roma

February 27, 2021

Abstract

The goal of the project was to implement a task-oriented speech dialogue system (SDS). Prototype of such a system for the mobile robot-helper to navigate, adapt, orient and get starting pack of the information for new Sapienza students has been implemented and will be described in the current report.

I. Functionality

THE aforementioned SDS (i.e. voice Rasa bot) is intended to be a build-in application to the mobile robot who is rolling out over assigned restricted area (University) and complete users' requests. Functionality of a bot primarily may be divided into two parts. One is a proposing of a user to get a subscription on the newsletter for newcomers, requesting user's email address and approval of a subscription. The second part provides the user with the timetable containing the information about timeslots, names of the professors and locations of the classes, if the student belongs to the Artificial Intelligence and Robotics group of users, and redirects on the Sapienza web page all other users. Intention of the users to perform a so-called bot-challenge also is taken into consideration.

II. Architecture

Acquired and installed tools are an open source machine learning framework for automated text and voice-based conversations Rasa, SpeechRecognition Python module (with its dependency PyAudio) that has been designed by Google, gTTS (Google Text-to-Speech) Python library and CLI tool to interface with Google Translate's text-to-speech API and mpg321 command-line mp3 player.

i. RASA

Abstracting from the speech-to-text and text-to-speech translations all of the functionality of the dialogue system is implemented using Rasa that is able to understand messages, hold conversations and connect to messaging channels and APIs.

i.1 Intents

For an assistant to recognize what a user is saying no matter how the user phrases their message, we need to provide example messages the assistant can learn from. We group these examples according to the idea or the goal the message is expressing, which is also called the intent. Intentions to be processed are described in the `nlu.yml` file.

Considered NLU-file of this project contains intentions: *greet*, *goodbye*, *bot challenge*, *pause*, *options*, *subscribe*, *inform*, *more info*, *agree*, *refuse*, *schedule*, *professors*, *rooms*.

Where *greet* and *goodbye* – standard initializers of the beginning and the end of dialogue, *bot challenge* – user's trials to ask for the identity of a robot, *pause* – thanksgiving and speech filler words, *options* – ask for the list of the available options, *subscribe* and *inform* – represent the intentions to get a subscription on the newsletter and provide with the information that is necessary for that, *agree* and *refuse* – standard sets of ways to express an agreement/disagreement (in the

current setting are employed exclusively for the distinguishing between AIRO/non-AIRO students, and intent *refuse* – for the provoking uttering of a goodbye in the case when user do not have any new requests), *more info* – intention to get access to the web page of the university (that is also used for the responds to non-AIRO students with the *schedule*, *professors*, *rooms* intentions – requests about the items corresponding to their names).

All of the intentions are provided with the 5 training examples.

i.2 Responses

The assistant needs responses it can send back to the user intents. In the **domain.yml** file of the Rasa project (or optionally in the **responses.yml**) bot's responses should be listed and given one or more (by the current design choice - 3) text options for each of them. If a response has multiple text options consistently to the Rasa dd-ones one of these options will be chosen at random whenever that response is predicted.

Performed configuration includes the following groups of responses: *utter greet*, *utter goodbye*, *utter options*, *utter ask request*, *utter i am a bot*, *utter ask email*, *utter subscribed*, *utter airo*, *utter more info*, *utter more info airo*.

In detail, *utter greet*, *utter goodbye* – responses to the intents of the same name, *options* – to represent functionality of the bot, *utter ask request* – asks for the request for the continuation of the conversation, *utter i am a bot* – bot-challenge handling, *utter ask email*, *utter subscribed* – collecting an information needed and confirmation of the completion of the subscription intention, *utter airo* – nesting point of the *schedule*, *professors*, *rooms* intent, *utter more info*, *utter more info airo* – nesting post-effects.

Note: *utter more info*, *utter more info airo* responses except for the textual fields contain also image fields with the links to the Sapienza web page and link to the Master's degree program in Artificial Intelligence and Robotics pdf-file correspondingly: [1] and [2].

i.3 Stories, Rules, Forms and Slots

Stories are example conversations that train an assistant to respond correctly depending on what the user has said previously in the conversation. The story format shows the intent of the user message followed by the assistant's action or response.

It is worthy to notice that apparent simplicity of the usage Rasa for the creation of the chat bot system is to be balanced by the main difficulty (i.e. drawback) of providing it with the number of the training examples sufficient for the effective training.

Newsletter subscription functionality interlined in a story where the user and assistant exchange greetings, the user asks to subscribe to the newsletter, and the assistant starts collecting the information it needs through the *newsletter form*.

Predefined scenarios in the form of stories are also intended to: react with the goodbye to the goodbye, with the self-identification information on the bot-challenge request, with the link to the Sapienza web page given explicit request for that, with an options list being asked about the options and with either again a link to the Sapienza web page or with the timetable pdf-sheet depending on who asks given a question about professors/rooms/times of the lessons.

Stories are defined in the separate **stories.yml** file that is stored in the data folder together with **nlu.yml** and **rules.yml**.

Rules bring a similar to the stories concept, but turn out to implement a more «hard» setting. They describe parts of conversations that should always follow the same path no matter what has been said previously in the conversation. Whenever we want our assistant to always respond to a certain intent with a specific action, we use a rule to map that action to the intent.

Generally saying developers of Rasa recommend to use rules as less as possible, but sometimes they are necessary for the realization of a some specific need, for example, in the case of dealing with forms. There are

many situations where an assistant needs to collect information from the user. For example, when a user wants to subscribe to a newsletter in our, the assistant must ask for their email address.

There are two rules one of which triggers the *newsletter form* whenever the user expresses the intent *subscribe* and the another one that triggers the *utter subscribed* action once all the required information has been provided. The second rule only applies when the *newsletter form* is active to begin with; once it is no longer active (*active loop: null*), the form is complete.

The form by itself contains just one field to collect an *email* and is included into **domain.yml** file (although this is also possible to keep them separately in the **forms.yml**) as well as slot that contains specification for the *email* field of the form.

The slot is a bot's memory intended to act as a key-value store which can be used to store information the user provided (or obtained as the result of a database query, but that is not the case here).

i.4 Actions

The mechanism is working in a following fashion: after each user message, the model will predict an action that the assistant should perform next. The list of actions is stored explicitly in the same **domain.yml** file and includes all of the possible responses except for those that are involved in the forms. So, in the considered setting list of actions is represented by *utter greet*, *utter goodbye*, *utter options*, *utter ask request*, *utter i am a bot*, *utter ask email*, *utter subscribed*, *utter airo*, *utter more info*, *utter more info airo* – all possible responses except for *utter ask email*.

ii. STT and TTS

This subsection of the report devoted to the speech-to-text and speech-to-text translations with the Google products integrated into the Rasa pipeline.

ii.1 External messages sending to the Rasa chat bot

To create a *speech* dialogue system with Rasa one has to initially send some voice note, and this voice note can not be directly sent to the Rasa chat bot. We need to know how to send messages to the Rasa dialogue system externally so that in the case of getting some voice as an input value, convert this input into the textual form which then will be indirectly sent to the chat bot in order to get output. In particular, we have to add an API on which post-request with respect to the message of the user is going to be sent.

For that **credentials.yml** and **endpoints.yml** should be set up appropriately, and then there is no need in any specific tool for running Rasa server except for the import of the **requests** Python module.

ii.2 Speech-to-Text

Speech Recognition is a process in which a computer or device record the speech of humans and convert it into text format. It is also known as Automatic Speech Recognition (ASR), computer speech recognition or Speech To Text (STT).

Module called **SpeechRecognition** that has been designed by Google is used to implement this part of a program as being known to be accurate and low-consuming in terms of the space of the system. **PyAudio** is also needed as a dependency for the aforementioned **SpeechRecognition** and for the further processing of the audio.

The voice is being carried out by the classes **Recognizer()** – to record (method *listen*) and convert (method *recognize_google*: Google is used over there (Google is accessing from the system given an Internet-connection) the audio, and **Microphone()** – to activate a microphone.

ii.3 Text-to-Speech

Rasa chat bot return the value in the textual format. The goal of Text-to-Speech part is to convert it into the form of a voice. Text-to-Speech (TTS) reads aloud digital text – the words on computers, smartphones and tablets.

In the current project **gtts** (Google Text to Speech) Python library is employed to implement a converting of the text to a voice and saving of this recording on the system. For the replaying of that sound media player is required. To this intent, command-line mp3 player **mpg321** has been chosen.

The **gTTS** class is to be imported from the library for the passing the text and language (specified as English, although English is set by default) to the engine. Then the method *save* is to be invoked for saving the converted audio in an mp3, whereupon we are only left with the playing the converted file through the *subprocess.call()*.

ii.4 Integration

Final step is to combine three parts described so far to work together as one just like a customer care executive, that is done in the **Voice_bot.py** script stored in the actions folder of the project.

III. Conclusion

The prototype of the students' mobile voice assistant bot had been created in this project based primarily on the Rasa framework and integration over it. Rasa helps to build contextual assistants capable of having layered conversations. In order for a human to have a meaningful exchange with a contextual assistant, the assistant needs to be able to use context to build on things that were previously discussed — Rasa enables to build assistants that can do this in a scalable way, however the key issue of such an approach – it needs to have tremendous number of training examples for the achieving a high performance – should be taken into account.

References

- [1] <https://www.uniroma1.it/it>
- [2] https://corsidilaurea.uniroma1.it/sites/default/files/orari_delle_lezioni_marr_1deg_e_2deg_anno.docx_0.pdf
- [3] Official Rasa web page with the Playground tutorials and documentation: <https://rasa.com/>

IV. Example dialogue

```
Your input -> H1
H1!
What do you want to ask me?
Your input -> Who are you?

If you want I will send you invitation information for the new students or display the link with the information about your time table. Also I can redirect you to the web page of the university
Your input -> Yes, but I asked who are you...

I am Sapienza helper bot
Your input -> A, okay

Let me know if you need something
Your input -> Send me an invitation information please

What is your Sapienza or personal mail box?
Your input -> fake@mail.com

Check your inbox at fake@mail.com in order to get invitation letter!
Your input -> Okay, thank you
```

```
Your input -> Okay, thank you

Let me know if you need something
Your input -> May be i want to scheck my schedule

Do you study Artificial Intelligence and Robotics?
Your input -> yes

I am sending you a link containing all information about your classes in this semester
Image: https://corsidilaurea.uniroma1.it/sites/default/files/orari\_delle\_lezioni\_marr\_1deg\_e\_2deg\_anno.docx\_0.pdf
Your input -> thanks

What do you want to ask me?
Your input -> nothing more

Have a nice day!
Your input -> bye

Goodbye
```