Artificial Intelligence [12 Credit]

Master in Artificial Intelligence and Robotics

- Prof. Nardi -

SHRI Project

Student: Francesco Cassini
Sapienza ID number: 785771
GITHUB directory: https://github.com/francesco-AI/AI-SHRI-Project.git

Abstract

For my Spoken Human-Robot Interaction project I chose to implement a robot gardener capable of performing a series of actions in the field of plant care and maintenance.

The chosen context allowed me to implement a series of dialogues through which to experiment with the use of speech-recognition and text-to-speech routines: through the "Google Speech Recognition" and "SAY" libraries it was possible to send and receive voice commands in my source language (that is to say: the Italian).

But in particular I stopped to study the StanfordNLP library for the treatment of the recognized text.

As for the semantic part, I studied the theoretical concepts that underlie the FRAME theory. I only used the FRAMENET library at the beginning of my project, when I still hadn't decided to implement everything in Italian.

Unfortunately Italian is not one of the languages supported by the current library and therefore, even having used it, it was not possible to insert it in a "thorough" way in this project, even if I took it into account at least as an approach in the recognition phase of the actions to accomplish.

Introduction

In order to have a more complete overview of the processes behind a Spoken Human-Robot Interaction, I have imagined how the dialogues between a robot gardener and a person could take place.

In particular, what would be the macro actions that a human being would like to be carried out by a robot and vice versa what is the information that the latter would have needed to complete his task.

The Map of LOCATIONS, PLANTS, TOOLS and of course our robot gardener

In order to simplify things, I have implemented a dynamic map of objects (both plants and equipment) that can be made to reach the robot.

```
#viene stampata il tavolo da gioco in cui composto da 100 o più caselle
def print_board(agent_position):
    fields = list(range(100))

    print("[A = Agent] [C = Casa] [S = Serra] [O = Orto] [G = Giardino]...")
    board = "-----------------------------------------\n"
    for i in range(0, 100, 10):
        delimiter = fields[i:i+10]
        for field in delimiter:
            if field == agent_position:
                board += "| A "
            elif field == casa1 or field == casa2 or field == casa3 or field == casa4 or field == casa5 or field == casa6 or field == casa7:
                board += "| C "
            elif field == serra0 or field == serra1 or field == serra2 or field == serra3 or field == serra4 or field == serra5:
```

After each command, the map will be updated showing the new location of both the agent and the equipment.

```
---
Gardener-bot say: Ciao, sono il tuo nuovo robot giardiniere
Gardener-bot say: Ti mostro la mappa della tua villa
[A = Agent] [C = Casa] [S = Serra] [O = Orto] [G = Giardino]...
-----------------------------------------
| C | C | C | C | C |   |   |   | S | S |
-----------------------------------------
| C | F | I | N | C |   |   |   | R | S |
-----------------------------------------
|   |   |   |   |   |   |   |   | R | S |
-----------------------------------------
|   |   |   |   |   |   |   |   | S | S |
-----------------------------------------
|   |   |   |   | A |   |   |   |   |   |
-----------------------------------------
|   |   |   |   |   |   |   |   |   |   |
-----------------------------------------
| G | G |   |   |   |   |   |   | O | O |
-----------------------------------------
| G | B |   |   |   |   |   |   | P | O |
-----------------------------------------
| G | B |   |   |   |   |   |   | P | O |
-----------------------------------------
| G | G |   |   |   |   |   |   | O | O |
-----------------------------------------
```

In particular, from the map we can see that there are 4 locations, 3 type of plants and 3 tools:

- **house**, which in turn contains 3 equipment:
  - **watering can (tool)**
  - **seeds (tool)**
  - **shears (tool)**

- **greenhouse**, where the **roses (plant)** are located

- **vegetable garden**, where the **tomato (plant)** plants are found

- **garden**, is formed by **bushes (plant)**

Three type of ACTIONS: Watering, Pruning, Sowing

Going into more detail in the example that I developed, I imagined that the user could ask in particular for 3 types of actions to be performed:

- ***water*** one of the three types of plants present in the villa

- ***prune*** one of the 3 plants

- ***sow / plant*** the seeds of a new plant

Implementation details

Technically speaking, the project uses 3 different libraries, but in the realization of the main idea I took inspiration from the theories that gave birth to a fourth FrameNet library that unfortunately it was not possible to use due to the absence of Italian among the supported languages .

Concept Idea (based on FrameNet theories)

"The FrameNet project is building a lexical database of English that is both human- and machine-readable, based on annotating examples of how words are used in actual texts. From the student's point of view, it is a dictionary of more than 13,000 word senses, most of them with annotated examples that show the meaning and usage." [1]

The idea behind FrameNet is that each word or verb can take on a different meaning depending on the context (or frame!) In which that word is used.
The FrameNet library is able to trace the meaning of a verb based on the words that surround it.
It is a library that we could define "complementary" to WordNet [3], that is, a lexical database that has become a standard on the internet.

At the beginning of my project I had tried to create an example that could understand the meaning of an "ambiguous" phrase and to perform the correct action based on what was understood.
Unfortunately, both the shortage of time and the use of the Italian language (FrameNet is not available in Italian *[2]*) prevented me to continue my study (which, however, I will not fail to carry on later).

In any case, the idea that I developed at the beginning was to be able to trace the meaning of a verb based on the words that surround it.

Something of the initial project remained, in fact the actions that the agent can carry out are not activated by a single verb, but by multiple types of verbs. Unfortunately it was not possible to implement more than this for now.

Speech-to-text (and viceversa)

There are various ways in which a person can interact with a robot, but in the case of the SHRI contest, the goal is to turn to a robot exactly as we would do with another person.

And since verbal communication is the one that distinguishes man from other animal species, what we want to achieve is that even a robot can communicate with man using his voice.

In recent decades there has been a constant development of Natural Language Programming techniques based on neural networks.

To do this, you need components in particular:
- **Speech-o-Text**: a voice recognition system which, given an incoming audio, transforms it into a text
- **Text-to-speech**: the inverse process, that is, given a text, a synthesizer transforms it into a spoken sentence.

The first part of the process is certainly the most complicated.
But we must not believe that the latter is no different.

I have used:

- **SpeechRecognition** 3.8.1 library [4]

```
def hear():
    r = sr.Recognizer()
    #r.energy_threshold = 200

    err = True
    while err:
        input("[Parla dopo aver premuto INVIO]")

        with noalsaerr() as n, sr.Microphone() as source:
            print("[Inizio Ascolto...]")
```

This library is made up of various components, including Google Speech Recognition and Google Cloud Speech API.

- **gTTS** 2.1.1 library [5]

```
def speak_simple(turn):
    tts = gTTS(text=turn, lang='it')
    tts.save("gardbot.mp3")
    print('\033[92mGardener-bot say: '+turn+'\033[0m')
    playsound.playsound('gardbot.mp3', True)
```

This library is a text synthetizer. It transforms a text into an mp3 that could be “play” from the OS system.

Stanford CoreNLP – Natural language software

"Stanford CoreNLP provides a set of human language technology tools. It can give the base forms of words, their parts of speech, whether they are names of companies, people, etc., normalize dates, times, and numeric quantities, mark up the structure of sentences in terms of phrases and syntactic dependencies, indicate which noun phrases refer to the same entities, indicate sentiment, extract particular or open-class relations between entity mentions.." [6]

Stanford NLP is a very powerful tool that permits a lot of possibility: from grammatical analysis tools, annotator and ran as a simple web service.

It is also available in many languages, including Italian. [7] [8]

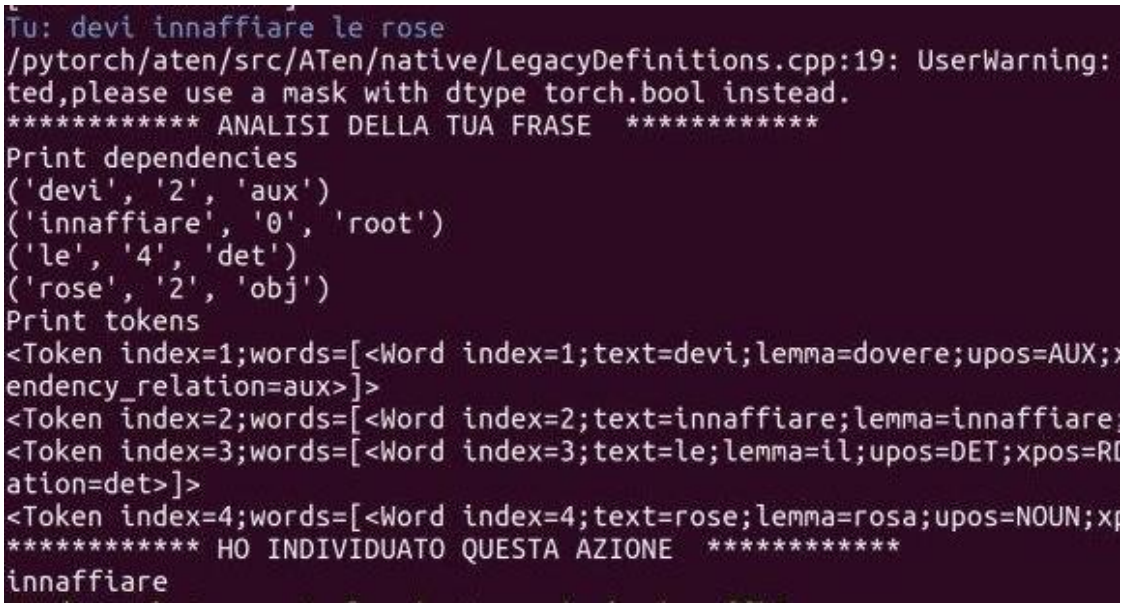
With the help of an example found online [9], I managed to extract a large number of features from the sentences captured through the speech recognition service.

How it works

Compared to other libraries (speech-to-text and text-to-speech), StanfordNLP is much more interesting to study and explain.
In fact, thanks to a series of "processors" it is able to analyze the sentences given as input and to bring out a series of very useful information for the purpose of the logical and semantic "understanding" of the text.

```
490             comando = hear()
491             doc = nlp(comando)
492             if debug_dependencies:
493                 print('************ ANALISI DELLA TUA FRASE  ************')
494                 print('Print dependencies')
495                 doc.sentences[0].print_dependencies()
496                 print('Print tokens')
497                 doc.sentences[0].print_tokens()
498
```

As we can see directly in the code, once the library has been called, the simple "nlp" command is enough to start the analysis of the text.
Then we can print and recognize in the text all the various parts of the sentence.



- ***Dependency extraction***: to get the dependency relations for all of its words

- ***Tokenization***: the token object contains the index of the token in the sentence and a list of word objects (in case of a multi-word token). Each word object contains useful information, like the index of the word, the lemma of the text, the pos (parts of speech) tag and the feat (morphological features) tag.[10]

In our case we found several information useful, in particular two:
- **the identification of the VERB**
- **the recognition of LEMMA** (ie the verb from which it was conjugated)

Thanks to these two information it was thus possible to identify the type of ACTION that the agent must perform.

Once the ACTION has been carried out (remember that there are 3), the user is always asked for other information through the same mechanism.

In particular, however, in each action, keywords are identified that the process expects for each action.
For example, if I have to move the agent, I need to know which of the 4 places he has to go.
Or on which of the 3 types of plants the action should be carried out.

Bibliography

[1] https://framenet.icsi.berkeley.edu/fndrupal/about

[2] https://framenet.icsi.berkeley.edu/fndrupal/framenets_in_other_languages

[3] https://wordnet.princeton.edu/

[4] https://pypi.org/project/SpeechRecognition/

[5] https://pypi.org/project/gTTS/

[6] https://stanfordnlp.github.io/CoreNLP/

[7] https://universaldependencies.org/treebanks/it_isdt/index.html

[8] https://stanfordnlp.github.io/stanfordnlp/models.html#human-languages-supported-by-stanfordnlp

[9] https://www.analyticsvidhya.com/blog/2019/02/stanfordnlp-nlp-library-python/

[10] https://www.analyticsvidhya.com/blog/2019/02/stanfordnlp-nlp-library-python/