**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-2021/

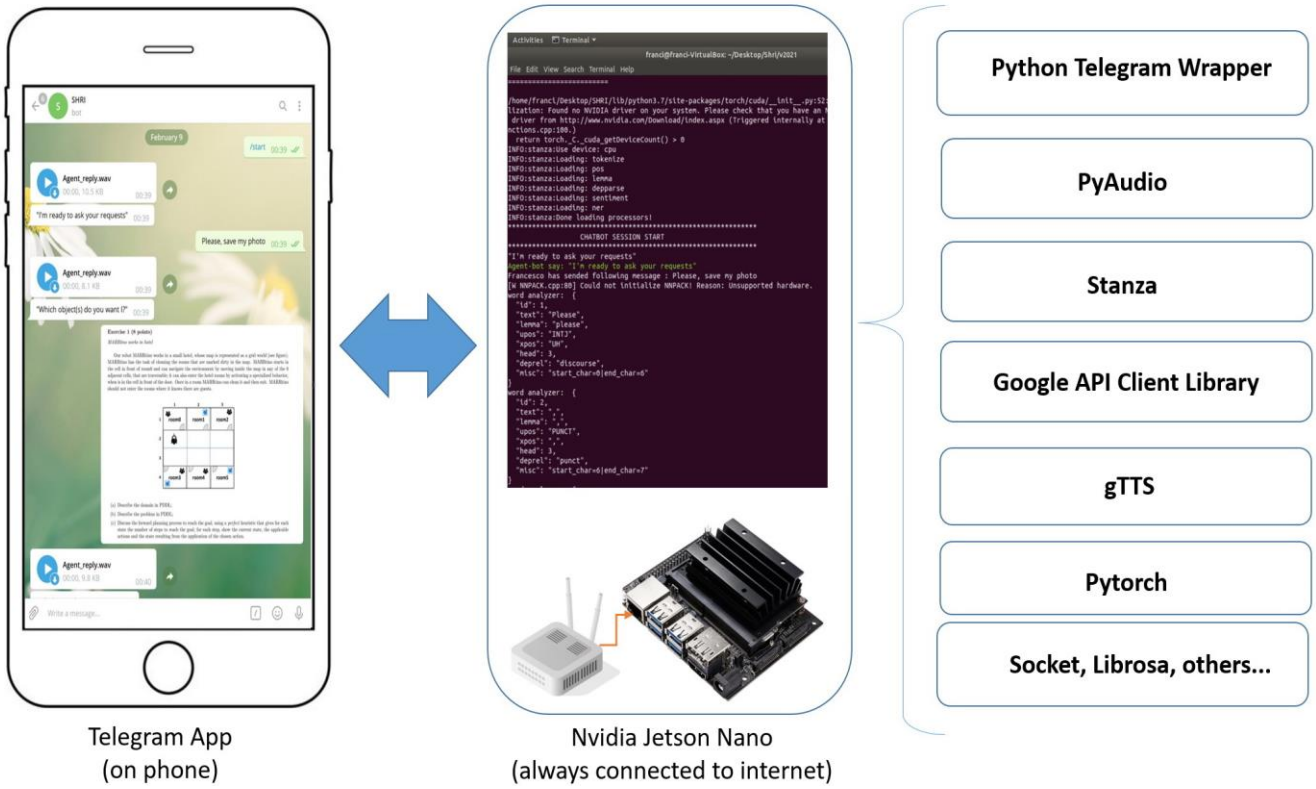
## Abstract

For my Spoken Human-Robot Interaction project I have implemented a bot that is capable of performing a series of actions through vocal commands via Telegram Bot API: one of the most interesting feature of my work is the control of a remote system. In fact I have installed it on a Nvidia Jetson Nano connected to internet to receive my commands in any moment of day.

From what concern the study of Dialogue manager, I focused my attention on the "intent recognition" block. In fact, I have implemented a sequential solution in which the "request" of the user is first analyzed by a "simple" circuit that uses the grammatical construct looking for matches in the dictionary of actions. If it fails, it passes the request to a neural network which is trained on the examples provided using the system. In this sense, the idea behind is to utilizes my system to collect specific data for the training of this second network (a very primitive sort of “continual learning”).

I was inspired by my daily use of a “Save folder” Telegram function, which allows to save material of interest (links, photos, files) in a special “folder”: it works like a “primitive” NAS (Network Attached Storage) system, but it's completely lacking of a “smart” search function. And it works like a memory attached to my bot. The chosen context allowed me to implement a series of dialogues through which to experiment the use of speech-recognition and text-to-speech routines: through the "Google Speech Recognition" and "SAY" libraries it’s possible to send and receive voice commands (in English).

## Introduction




Telegram App
(on phone)

Nvidia Jetson Nano
(always connected to internet)

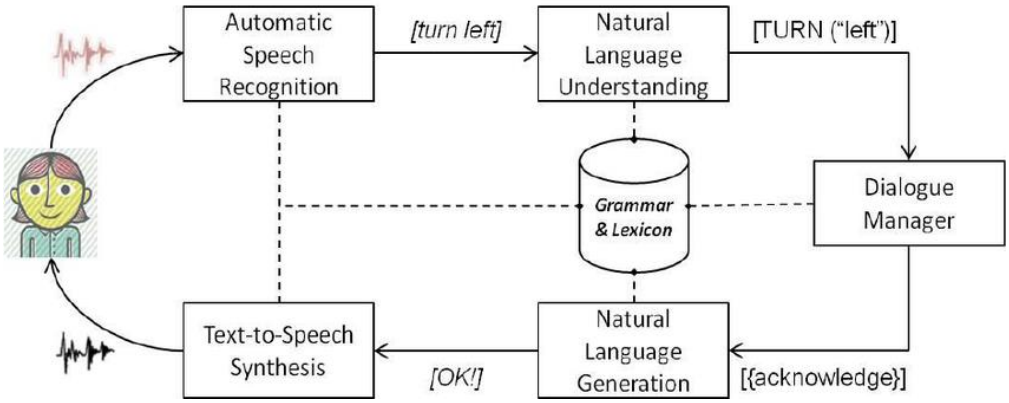
The best way to make things work is to use them every day in order to find flaws and correct them.
In order to implement this simple principle, I have thought about creating a platform that allows me to use and improve a bot-agent running on an “energy-saving remote server” like the Nvidia Jetson Nano.

This project will not end with its presentation, but will be further pursued over the next months: the functions currently present are all oriented to future expansion of the platform.

In fact, the system is not only able to receive, archive and extract files and documents from a database, but also to work as:
- a "dataset creator", thanks to the possibility of tagging and saving all the phrases it receives with the minimum effort.
- add new functions to its task database

The heart of the project is to build a Dialogue manager that, using different neural networks, is able to plan increasingly complex sequences of actions.

## Project Architecture and Implementation details




The project architecture is "classical":
**1 step - ASR**: the input phrase are converted in text through Google API speech recognition
**2 step - NLU**: text is analyzed by Stanza (it's the new nlp library from Stanford NLP group)
**3 step - DM**: find the next action based actual state
**4 step - NLG**: generation of response
**5 step - TtS**: conversion of text in speech (thorugh Google text to speech synthesis)

### *Step 1 and 5: ASR and TTS*

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 talk to a robot exactly as we would do with another person.

To do this, we need some particular components such as:
- **Speech-to-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.

To realize ASR, I have used:

**SpeechRecognition** 3.8.1 library [4]

```
def hear_uservoice(update: Update, context: CallbackContext) -> int:
    global Agent, User
    user = update.message.from_user
    audio_file = update.message.voice.get_file()
    audio_file.download('audio.wav')

    x,_ = librosa.load('audio.wav', sr=16000)
    sf.write('tmp.wav', x, 16000)
    r = sr.Recognizer()
    with sr.AudioFile('tmp.wav') as source:
        audio = r.record(source)
    try:
        s = r.recognize_google(audio)
        #comando = r.recognize_google(audio_file, language="en-EN")
        comando = s
    except Exception as e:
        print("Exception: "+str(e))
```

This library is made up of various components, including Google Speech Recognition and Google Cloud Speech API. Telegram has a function to send voice messages: so in this case we have only to take the audio.wav, convert it send to recognizer.

**gTTS** 2.1.1 library [5]

```
### SIMPLE ACTION
def speak_simple(text):
    tts = gTTS(text=text, lang='en')
    tts.save("Agent_reply.wav")
    print('\033[92mAgent-bot say: '+text+'\033[0m')
    playsound.playsound('Agent_reply.wav', True)
```

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

## *Step 2: Stanza (before it was Stanford CoreNLP)*

*“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..” [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]

**How it works**

Stanza/StanfordNLP 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.

```
/pytorch/aten/src/ATen/native/LegacyDefinitions.cpp:19: UserWarning:
ted,please use a mask with dtype torch.bool instead.
************ ANALISI DELLA TUA FRASE  ************
Print dependencies
('devi', '2', 'aux')
('innaffiare', '0', 'root')
('le', '4', 'det')
('rose', '2', 'obj')
Print tokens
<Token index=1;words=[<Word index=1;text=devi;lemma=dovere;upos=AUX;x
endency_relation=aux>]>
<Token index=2;words=[<Word index=2;text=innaffiare;lemma=innaffiare
<Token index=3;words=[<Word index=3;text=le;lemma=il;upos=DET;xpos=RD
ation=det>]>
<Token index=4;words=[<Word index=4;text=rose;lemma=rosa;upos=NOUN;xp
************ HO INDIVIDUATO QUESTA AZIONE  ************
```

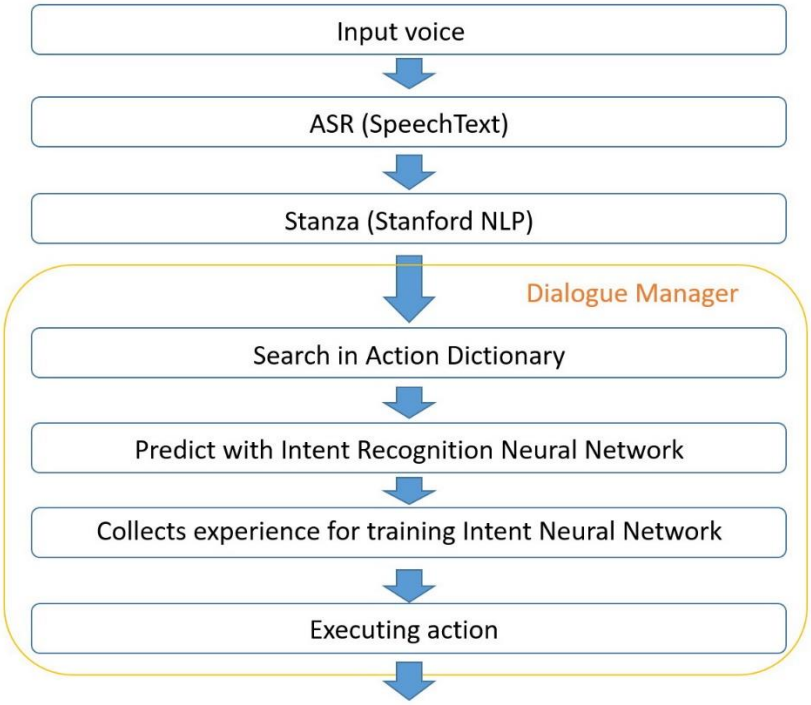
- ***Dependency extraction***: to get the dependency relations for all of its words

- ***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]

## Step 3 - Dialogue manager

A Dialog Manager (DM) is a component of a Dialog System (DS), responsible for the state and flow of the conversation. Based on speech act theory, we can describe every speech act as an action that "influence" the state of the agent. Dialogue manager tracks the state of agent, deciding which is the next action of agent.

The first part of DM is the "intent recognition" through the analysis of the phrases and the search in a database of actions.
This part is perhaps the most important because it "addresses" the behavior of the system.




I have focused my work in the improvement of this part other than for this project also for my personal interesting.

How it works the pipeline of "intent recognition"?
I have implemented a "double circuit" with the scope to make an initial "prediction" with a lexical analysis and at same time to collect data.
The idea is to use a "simple" neural network (I have converted a "sentiment analysis" multiclass network realized in pytorch) trained on phrases given in input during the daily utilization of the system.

**Description of Intent Recognition phase**
<u>First step</u>
Stealing some concept from FrameNet theory, the first step of my intent recognition block is the identification of the verb in the phrase and then checking if it corresponds to some action in the action dictionary.

*"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.

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. More, FrameNet library unfortunately doesn't include Italian among the supported languages. And in my future project I would to create a multilingual environment.

How is realized this part?
Through Stanza is possible to analyze the phrase received in input after ASR pipeline and extract the grammar and lexical information, 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.

<u>Second step</u>

One of the problem of an intent recognition block based on grammar and lexical analysis is the "rigid" functioning: in fact, if a word is not inserted in the database, the system will certainly not recognize it or in any case it will struggle to distinguish between several meanings.

In this sense, deep learning neural networks, especially those based on Transformer and Bert, are able to approximate the meaning of a sentence much more accurately.

The problem with neural networks, however, is the quantity of samples to be given to the network in order to train them to distinguish between several different meanings.

So how to solve this situation?

I'm trying to solve it using a "double step of analysis":
- the first step analyzes phrases and ask user if its analysis is correct
- in case negative, the system asks to user the correct label of the phrase and collect the phrase and the label in a dataset
- after to have collected a sufficient number of samples, it will be possible train a deep learning network that will make intent recognition more accurate.

Over time the first circuit will become useless. Moreover, it will always be possible to insert sentence analysis routines inside the neural network to help it manage the recognition phase.

Future development

As I said, I want to further develop this project:
- extend multilingual support through a more complex neural architecture with Bert and Transformer, trained on MNLI
- implement a memory augmented neural architecture to save samples and objects
- redesign the action mechanism pipeline also in the sight of a planning circuit based on a Reinforcement learning

## 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://github.com/python-telegram-bot/python-telegram-bot