UNIVERSITÀ DEGLI STUDI
DI GENOVA

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

Enhancement of Pepper Abilities - Project Report

Alberto Ghiotto (RobEng), Amrita Suresh (EMARO+)
Francesca Canale (RobEng), Tommaso Gruppi (RobEng)
Supervisors: Antonio Sgorbissa, Carmine Recchiuto, Roberto Menicatti

March 5, 2020

Abstract

This report describes our approach to define an activity that listens for the user request, searches Wikipedia and returns information from the page it finds. 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

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, such as playing music, setting a reminder, reading the news of the day, and many others, but can also talk about general conversation 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 principle, the user could talk about something that is not encoded in the ontology, and in this case the robot will reply by stating that it has probably not understood what the user said.

The 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.

2 Software Tools

In this project, we have implemented all modules using the Python programming language. In order to run these modules on a remote PC and send commands to Pepper, we use the **Python NAOqi SDK** [**p1**]. The NAOqi API provides several Python modules to communicate and control Pepper, for applications such as speech recognition, motion, perception and interaction. The **wikipedia** API allows us to search and extract information from any page on Wikipedia using several functions. In the next section, we have explained how we use these tools to solve the problem being discussed.

3 Solution Approach

Figure 1 shows the expected sequence of events and actions, along with the modules that are used to perform them.

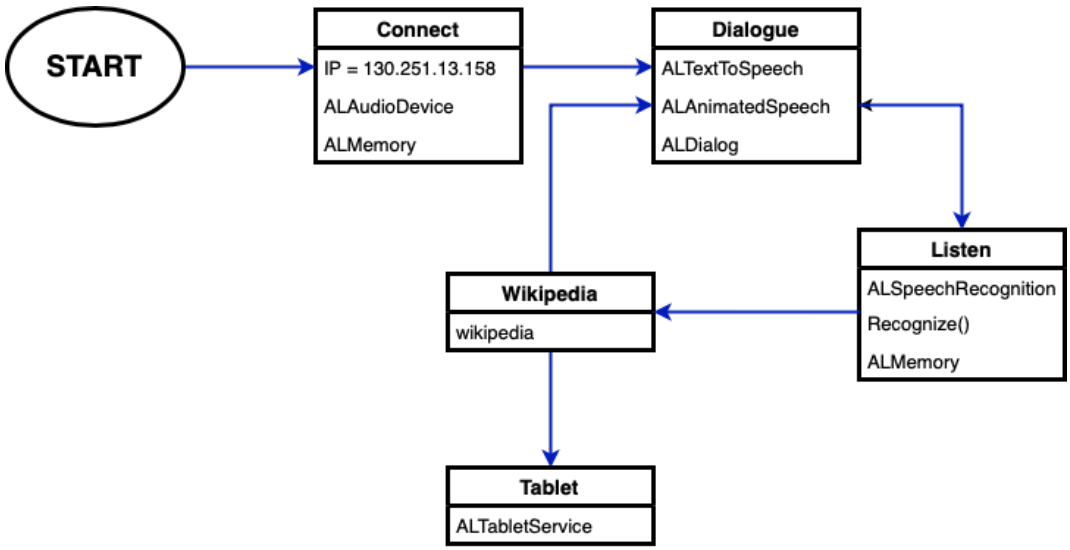

Figure 1: Actions performed and the modules used for each action

We have used two Python modules, namely, *helloWorld.py* (Appendix A) and *prova.py* (Appendix B). A short description of what each module does is as follows:

- **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.

- **ALAnimatedSpeech** : To create more lively speech by annotating the text.
- **ALDialog** : Used when a dialog needs to be read from a .top file.
- **ALTabletService** : Displays images and videos on the tablet.
- **ALSpeechRecognition** : Facilitates human speech recognition.
- **Python wikipedia** : Library used to access and parse data from Wikipedia.
- **Recognizer** : Recognizes user input using Google Speech Recognition. Defined in B.

According to figure 1, the different actions performed at each stage of interaction are *Connect*, *Dialogue*, *Listen*, *Wikipedia* and *Tablet*. How this is implemented has been explained below:

- **Connect** : At startup, we connect to the Pepper robot using its IP address. We then connect to the relevant modules at port **9559** using the *ALProxy* module of the NAOqi API.
- **Dialogue** : We have created a local vocabulary of words for Pepper which is dynamically updated as the conversation continues. This vocabulary object is helpful when we want to allow the user to speak in natural language instead of a narrowly guided conversation. The dialogue between the user and the robot proceeds according to a predefined script:

  PEPPER: What would you like to know about?
  USER: *<search keyword for wikipedia>*
  PEPPER: *<read page content if found and display image on tablet>*
  Do you want more information?
  USER: *<Yes or No>*
  PEPPER: *<If yes>* Great! Which one of the following topic would you like to know more about? *<Read table of contents>*
  <If no> Ok.
  <Else> I'm sorry I didn't get that. Please answer yes or no.
  USER: *<Subtopic name>*
  PEPPER: *<Go to subtopic and read contained information>*

  To carry out the above script, we use the **say** command from the ALTextToSpeech module to make the robot speak.
- **Listen** : The robot listens for user responses using the **listen** command defined in the Recognizer class in B. When an input sentence is recorded, we first check if the words (all or keywords) already exist in the vocabulary. Then we respond accordingly with the next step or dialogue.
- **Wikipedia** : After extracting the keywords from the user input, Pepper searches wikipedia for the these keywords and reads out information from the page found. The **wikipedia** module defines several objects such as **page**, **sections**, etc. to extract and use information from any Wikipedia page.

- **Tablet** : **showImage** and **hideImage** commands from the ALTablet module are used to display images on Pepper's tablet.

4 Work to be done

- *contents* list object is empty. Possible bug in the wikipedia library.
- Incorporation of robot gestures during the conversation.
- To make the conversation as natural as possible. An efficient keyword recognition algorithm needs to be written, possibly using Google Speech Recognition modules.

5 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/`

Appendices

A helloWorld.py

```
import naoqi
from naoqi import ALProxy
import wikipedia
import unicodedata # To convert unicode (read from wikipedia) to string
import time
from prova import Recognizer
import sys
import functools
import argparse


IP = "130.251.13.158"

# Manage audio inputs and outputs, it is used by all other audio modules
audio_module = ALProxy("ALAudioDevice", IP, 9559)
mem_module = ALProxy("ALMemory", IP, 9559)

# Connects to speech proxy to make the robot speak
speak_module = ALProxy("ALTextToSpeech", IP, 9559)

```

```
# Connection to ALAnimatedSpeech proxy: it can create more lively speech by annotating the text yourself with some instructions or by adding animations for some words with ALSpeakingMovement
animated_module = ALProxy("ALAnimatedSpeech", IP, 9559)

# Connects to movement service to give the autonomous ability while speaking(enabled by default)
# movement_module = ALProxy("ALSpeakingMovement", IP, 9559)

# Connects to tablet proxy to display image
tablet_module = ALProxy("ALTabletService", IP, 9559)

# Connect to dialogue proxy, we need to create and upload a .top file
dialogue_module = ALProxy("ALDialog", IP, 9559)
dialogue_module.setLanguage("English")  # Set the language

# Connects to speech proxy to make the robot understand what a human says
understand_module = ALProxy("ALSpeechRecognition", IP, 9559)
understand_module.setLanguage("English")  # Set the language

# Define the keyword
keyword = "Barack Obama"
vocabulary = ["yes", "no"]
vocabulary.append(keyword)

# Load and access data from full Wikipedia pages
ny = wikipedia.page(keyword)
imageStr = unicodedata.normalize('NFKD', ny.images[2]).encode('ascii','ignore') # To get the image
content = unicodedata.normalize('NFKD', wikipedia.summary(keyword, sentences=1)).encode('ascii','ignore') # To read the first phrase
sections = ny.sections  # To get the name of the sections
for i in sections:
    print i

print sections

# Add the sections into the vocabulary
vocabulary.extend(sections)  # We use extend to append a list to another list
understand_module.setVocabulary(vocabulary, False)
speak_module.say(keyword)

# Display the image on the tablet
tablet_module.showImage(imageStr)
time.sleep(5)
tablet_module.hideImage()

# Say the summary
speak_module.say(content)

time.sleep(1)

# Starting the recognizer
rec = Recognizer(IP)
```

```
70
71 while True:
72     speak_module.say("Do you want more information?")
73
74     # Listening for the answer
75     user_input = rec.listen()
76     rec.cleanMemory()
77     print user_input
78
79     if user_input == "yes":
80         speak_module.say("Great! Which one of the following topic
    would you like to know more about?")
81         for i in sections:
82             speak_module.say(i)
83             print i
84
85         # Listening for the answer
86         user_input_section = rec.listen()
87         rec.cleanMemory()
88         print user_input_section
89
90         section_summary = unicodedata.normalize('NFKD', wikipedia.
    summary(ny.section(user_input_section), sentences=1)).encode('
    ascii','ignore')
91         break
92
93
94     elif user_input == "no":
95         speak_module.say("Ok.")
96         break
97
98     else:
99         speak_module.say("I'm sorry I didn't get that. Please
    answer yes or no.")
100         pass
101
102
103 # understand_module.subscribe("WordRecognized")
104 # # speechRecognized = mem_module.subscriber("WordRecognized")
105 # stopped = False
106
107 # try:
108 #     while not stopped:
109 #         word = mem_module.getData("WordRecognized")
110 #         if len(word) > 0:
111 #             if word[0] == "yes":
112 #                 speak_module.say("Great! Which one of the
    following topic would you like to know more about?")
113 #                 speak_module.say(sections)
114 #                 stopped = True
115 #             elif word[0] == "no":
116 #                 speak_module.say("Ok.")
117 # except KeyboardInterrupt:
118 #
119 #     sys.exit()
120
121 # def onSpeechRecognized(msg, value):
122 #     print type(value)
```

```
#      print type(value[0])
#      if value[1] > 0.4:
#          got_keyword = value[0][6:-6]

# id_speechRecognized = speechRecognized.signal.connect(functools.
    partial(onSpeechRecognized, "WordRecognized"))
```

Appendices

B prova.py

```
import qi
import sys
import time

class Recognizer():
    def __init__(self, ip):

        try:
            # Initialize qi framework.
            self.session = qi.Session()
            self.session.connect("tcp://" + ip + ":" + str(9559))
            print("\nConnected to Naoqi at ip \"" + ip + "\" on
    port " + str(9559) + ".\n")

        except RuntimeError:
            print ("Can't connect to Naoqi at ip \"" + ip + "\" on
    port " + str(9559) + ".\n"

                                    "Please check your script arguments
    . Run with -h option for help.")
            sys.exit(1)

        self.pMemory  = self.session.service("ALMemory") #,
    "130.251.13.158", 9559)
        self.pASR     = self.session.service("ASR2") #,
    "130.251.13.158", 9559)
        self.speech_reco_event = "Audio/RecognizedWords"

        self.initGoogleRecognition()

    def initGoogleRecognition(self):
        '''
        Subscribe to the microevent self.speech_reco_event, which
    is raised by Softbank Robotics' AbcdkSoundReceiver.py
        inside the _processRemote() function whenever a sentence is
     recognized by Google services.
        '''
        ## If the action previously ended in a wrong away, clean up
     the memory
        self.cleanMemory()
        self.pMemory.subscriber(self.speech_reco_event)

    def cleanMemory(self):
```

```
        '''
        Clean Pepper's memory from the previously detected words/
    sentences.
        '''
        try:
            self.pMemory.insertData(self.speech_reco_event, [])
        except:
            pass

    def listen(self):
        '''
        Get the user input from Pepper's memory, recognized through
     Google Speech Recognition.
        Clean the memory right before returning, otherwise the same
     input will be returned every time this function is
        called even if the user did not talk anymore.
        '''
        self.pASR.startReco("English", False, True)
        _timeout = 15
        start_time_no_input = time.time()
        start_time_silence = time.time() + 3600

        user_input = ""
        input = None
        input = self.pMemory.getData(self.speech_reco_event)

        while True:
            while input == None or input == []:
                if time.time() - start_time_silence > 0.1:
                    return user_input
                if time.time() - start_time_no_input > _timeout:
                    return None

                input = self.pMemory.getData(self.speech_reco_event
    )

            sep = " " if not user_input == "" else ""
            user_input = user_input + sep + input[0][0].decode('utf
    -8')
            start_time_silence = time.time()
            start_time_no_input = time.time()
            self.cleanMemory()
            input = self.pMemory.getData(self.speech_reco_event)

        self.pASR.stopReco()
```