# MKI59: Robotlab practical

### Practical 4: Speech and Wit.ai

December 7, 2017

## 1 Introduction

The last few lessons, we explored various methods to capture and/or process data from the Nao sensors, like touch/sonar/speech/camera. In the previous lesson we explored processing of streams of camera images, continuing on with the camera.

In this lesson we will explore the capturing and processing of speech fragments. Compared to lesson 2, where we used the built-in speech processing capacities of the NAO, today we will go a little bit further out-of-the-box. We will try to process speech data outside the robot, by capturing audio from the NAO and forwarding it to the ASR server residing at `Wit.ai` (ASR=automated speech recognition)

First, we will do a short recap on what a self made module should look like and what features are mandatory in order to process events and register the module in NaoQi.

### 1.1 Modules: Recap

There are some important concepts when making your module, hereby a list with things you should not forget!

- Every method should have a docstring, starting with """
- The name of the module should be the same as the variable name.
- Make sure you have failsafe code in your initialization.
- Make sure your module inherits from ALModule
- Make sure when initializing your module, your module is a python 'global' variable.
- Make sure to 'exit' your module and to unsubscribe for events.

Code snippet 1: Failsafe for module creation.

```
#this codesnippet should be first in the initialization.
try:
    p = ALProxy(name)
```

```
    p.exit()
except:
    pass
```

## 2 Working with external speech processing

The audio module we will implement records audio for three seconds after which it transmits the audio file to Wit.ai. The main method looks like this:

Code snippet 2: Main method of Wit processing

```
global NaoWit

if __name__ == "__main__":
    pythonBroker = ALBroker("pythonBroker", "0.0.0.0", 9600, ip, port)
    #Use a UNIQUE name for the variable and string, such as your name
    NaoWit = NaoWitSpeech("NaoWit")
    NaoWit.startAudioTest(3)
```

Note that the name of the variable /textitNaoWit has to be unique, and the string given to NaoWitSpeech has to be the same as the variable name.

In the initialization of the module we make a proxy to ALAudioDevice and we set the parameters for the audio stream. Know that these parameters are very important, because this configuration is the only configuration that works with Wit.ai. We will be saving the incoming stream to a temporary file, for which we use the python module StringIO.

Code snippet 3: Initialization of the module

```
class NaoWitSpeech(ALModule):
    def __init__(self, name):
        """For writing audio to an external service"""
        ALModule.__init__( self, name );
        self.saveFile = StringIO.StringIO()
        self.ALAudioDevice = ALProxy("ALAudioDevice")
        channels = [0,1,0,0]
        self.ALAudioDevice.setClientPreferences(self.getName(), 48000, channels, 0, 0)
```

*Now, look back at assignment 2 to extend the initialization method with fail safe structures and import the right modules*

Two processing methods are required, one is *process* and the other is *processRemote*. When running your code from the computer the NAO will send the data to *processRemote*, if you run the code directly on the NAO (through a shell) the NAO will call *process*.

Code snippet 4: processRemote method

```
def processRemote(self, nbOfInputChannels, nbOfInputSamples, timeStamp, inputBuff):
        """
        Incoming method for data, writes it to savefile. For remote use.
        Note that docstring is mandatory!
        """
        self.saveFile.write(inputBuff)
```

*Extend the module with both processRemote and process (hint: it has the same arguments)*

The next method sets the headers for the request to the wit.ai server. The authentication code for Wit.ai can be found in the code snippets. It is advised to get your own authentication key from Wit.ai, by logging into their site `wit.ai`:

Code snippet 5: Setting http headers for Wit.ai

```
def startWit(self):
        """
        Method for creating the call to wit.ai.
        """
        self.headers = {'Authorization': 'Bearer 4KQMHH5QWJDMD7QSOM4RLDRFJ7Q6HALP'}
        self.headers['Content-Type'] = 'audio/raw;encoding=signed\
                        integer;bits=16;rate=48000;endian=little'
```

The next step is to start the audio recording and to upload this recording. You will see that, when implementing this code, you will not get any feedback when the robot starts recording.

Code snippet 6: Start recording & Uploading label

```
def startAudioTest(self, duration):
        """
        Subscribe for audio data.
        """
        self.startWit()
        self.ALAudioDevice.subscribe(self.getName())
        time.sleep(duration)
        self.ALAudioDevice.unsubscribe(self.getName())
        self.startUpload(self.saveFile)

def startUpload(self, datafile):
        """
        Start upload of speechfile.
        """
        conn = httplib.HTTPSConnection("api.wit.ai")
        conn.request("POST", "/speech", datafile.getvalue(), self.headers)
        response = conn.getresponse()
        data = response.read()
        self.reply = data
        print data
```

Now we have captured an audio (speech?) fragment, we will develop the processing part of this module.

*Make sure the robot plays a sound before and after the recording. You can use files directly on the NAO (`/usr/share/naoqi/wav/end_reco.wav` and `/usr/share/naoqi/wav/begin_reco.wav`). Take a look in the Aldebaran API on how to play sounds on the NAO.*

Code snippet 7: The following imports you might wanna make...

```
import httplib, time, StringIO
```

## 3 Interpreting the wit.ai response

The data returned by the wit.ai recognizer is in *json* format. Typically, it contains some fields including the recognized text and the corresponding confidence value. The following script uses the *json* package to parse a typical json object returned by wit.ai:

Code snippet 8: Example of accessing the wit.ai response

```
import json

wit_response = NaoWit.reply
resp = json.loads(wit_response)

conf = resp["outcomes"][0]["confidence"]
txt  = resp["outcomes"][0]["_text"]
print conf
print txt
```

Code snippet 9: Example structure of a json object

```
wit_resonse = """
{
    "msg_id" : "387b8515-0c1d-42a9-aa80-e68b66b66c27",
    "_text" : "how are you doing today robot",
    "outcomes" : [ {
        "_text" : "how are you doing today robot",
        "intent" : "query_metrics",
        "entities" : {
            "metric" : [ {
              "metadata" : "{'code' : 324}",
              "value" : "metric_visitor"
            } ],
            "datetime" : [ {
                "value" : {
                    "from" : "2014-07-01T00:00:00.000-07:00",
                    "to" : "2014-07-02T00:00:00.000-07:00"
                }
              }, {
                "value" : {
                    "from" : "2014-07-04T00:00:00.000-07:00",
                    "to" : "2014-07-05T00:00:00.000-07:00"
                }
              } ]
        },
        "confidence" : 0.621
    } ]
  }
"""
```

*Try this script to output the confidence value and recognized text for your spoken input. See if you can correlate (by observation) low confidence values to mis-recognized speech. Make your NAO speak out the recognized speech. Try to handle the error you get when no speech is received.*

## 4 Assignment

We have provided you with this brief tutorial because the NAO speech recognition capabilities are limited. Wit.ai is recognized as a state of the art recognizer that can handle "natural language". So, where the NAO can recognize isolated words, with Wit.ai you can recognize whole sentences. For this assignment, you will develop a speed-dating robot. The speed dating session takes 2 minutes during which the robot should ask some questions and attempt to understand the responses. No "yes/no" kind of questions should be used! The interaction should be as natural as possible – add gestures/head motion/face recognition etc. to spice up the interaction. You want your NAO to make the best first impression possible! I will score the human-likeness and naturalness of the conversation (i.e., how many gestures the robot uses, and how they correspond to the recognized words/sentences, how you handle miscommunications etc.)