# Audio & sonar

**MKI59: Robotlab practical**
**November 14, 2018**

In this tutorial we will look at how to recognize speech commands and how to work with the sonar.

## 1 Audio: Subscribing to speech recognition events

We will now write our own speech recognition module, which can be used to subscribe to recognized words and to react on these events by calling your own callback method.

Code snippet 1: Speech recognition module

```
import time
import sys
import naoqi
from naoqi import ALModule, ALProxy, ALBroker

ip = "192.168.1.102"
port = 9559

global memory
memory = ALProxy("ALMemory", ip, port)


class SpeechRecognition(ALModule):
    def __init__(self, name):
        try:
            p = ALProxy(name)
            p.exit()
        except:
            pass
        ALModule.__init__(self, name);
        self.response = False
        self.value = []
        self.name = name
        self.spr = ALProxy("ALSpeechRecognition")

    def getSpeech(self, wordlist, wordspotting):
        self.response = False
        self.value = []
        self.spr.setVocabulary(wordlist, wordspotting)
        memory.subscribeToEvent("WordRecognized", self.name, "onDetect")

    def onDetect(self, keyname, value, subscriber_name):
        self.response = True
        self.value = value
        print value
        memory.unsubscribeToEvent("WordRecognized", self.name)
        self.spr.pause(True)
```

This new module class extends `ALModule`. When writing your own module, please consider that this mechanism is standard for every module within the Nao. For detecting speech we need two methods and an initialization. In the initialization we define `memory` (subscribing to events) and a proxy to the speech recognition module. By subscribing to an event, also the event name, name of the module and a name for the callback method are required.

In this case the callback method is `onDetect()`. When an event is triggered, `onDetect()` is called with the event name, value and the name of the subscriber. We do not subscribe for events in the initialization, because we want to control when to listen for words. The `getSpeech()` method is used for subscribing to events. We also need to give a list with words and a boolean for whether the task is to recognize separate words or word spotting.

Finally, we need a main function to be able to test our module.

Code snippet 2: main function for module

```
if __name__ == "__main__":
    pythonBroker = ALBroker("pythonBroker", "0.0.0.0", 9600, ip, port)
    Speecher = SpeechRecognition("Speecher")
    Speecher.getSpeech(["cookie", "robot", "yumyum"], True)
    while Speecher.response is False:
        time.sleep(1)
```

The `Broker` concept in object-oriented and distributed programming may or may not be familiar to you. It is less important for the current exercises.

*Take the code snippets and write this module for speech detection / recognition. Let the robot ask a yes or no question and vary the response based on the answer. Make the Nao respond to control commands, like "lift head", or just "move left" or "move right" or "stop".*

Please note the delicate difference between subscribing and unsubscribing to events in the **Tactility** tutorial and speech (this section) examples. This is quite well-known in interactive applications: detecting when something has happened should result in immediate and appropriate reactions (and not in repeatedly responding to something that is already known).

## 2 Working with Sonar

The Nao robot has two sonar emission/detection pairs on its chest. These sonars are part of the `ALSonar` module, to which we can subscribe. It is important to know that we first subscribe to this module, before we extract any values from the memory. We can work with the sonar in two ways. The first way is given in the next example and concerns reading raw sonar values using the `getData()` method.

Code snippet 3: Extracting sonar values from memory

```
memory = ALProxy("ALMemory", IP, PORT)
sonar = ALProxy("ALSonar", IP, PORT)

sonar.subscribe("myS")
print memory.getData("Device/SubDeviceList/US/Left/Sensor/Value")
print memory.getData("Device/SubDeviceList/US/Right/Sensor/Value")
```

```
sonar.unsubscribe("myS")
```

*Take this code snippet and run it on your robot. Explore the raw sonar data, what are the perception boundaries? Is this an efficient way of working with sensor data?*

A more convenient way to work with the sonar is to subscribe to events (as we did in the previous sections), see http://doc.aldebaran.com/2-1/naoqi/sensors/alsonar.html#alsonar. There are four events which are generated, once we subscribe to the memory:

- `SonarLeftDetected`
- `SonarRightDetected`
- `SonarLeftNothingDetected`
- `SonarRightNothingDetected`

By subscribing to one or more of these keys, we can work with the sonar the same as we worked with face recognition in the previous example (react upon an event). Be aware that the threshold is fixed at 50 cm. This involves that events associated with object detection only occur when objects are (about) in front of the Nao and closer than 50 cm.

When there is no obstacle, `SonarLeftNothingDetected` and `SonarRightNothingDetected` are raised at the same time. This means that there is nothing in front of the Nao.

Given below is part of the module and the event handler.

Code snippet 4: Part of sonar event handling

```
global motionProxy

class SonarTut(ALModule):
    def __init__(self, strName):
        try:
            p = ALProxy(strName)
            p.exit()
        except:
            pass
        self.front = False
        ALModule.__init__( self, strName );
        self.memory = ALProxy("ALMemory")
        sonard = ALProxy("ALSonar")
        sonard.subscribe("SonarTut")
        self.memory.subscribeToEvent("SonarLeftDetected", strName, "sonarLeft")
        self.memory.subscribeToEvent("SonarRightDetected", strName, "sonarRight")

    def sonarLeft(self, key, value, message):
        motionProxy.stopMove()
        print "Something on the right. Stop moving."
```

*Make the Nao walk and account for objects in the same time. Let it stop when an object is detected on both sonars (e.g. a wall). You can use move(veloX, veloY, theta) from ALMotion and stopMove().*

What you probably noticed, is that the 50 cm threshold actually works quite robustly. . . if the Nao (or approaching object) do not move too quickly.

*Combining the raw sonar data (detecting objects beyond 50 cm) and these events may result in a robot that (i) walks at a fast pace; (ii) detects (using raw sonar data) that some object is* *approaching; (iii) uses the pre-cooked sensor events which indicate that an object is near; and (iv) reacts appropriately to this knowledge.*