MKI59: Robotlab practical

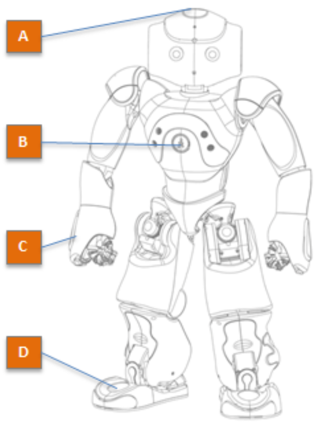
Practical 2: NAO Sensors Overview

November 23, 2017

In this practical we are going to work with a few of the NAO robot sensors. The Aldebaran website (`http://doc.aldebaran.com/2-1/family/nao_dcm/actuator_sensor_names.html`) provides an elaborated overview of the actuators (i.e., motor positions, voltage, temperature, stiffness) and sensors (i.e., touch sensors, sonars, inertial sensors). We will (1) illustrate the general set up of reacting on values via the touch sensors, (2) work out examples to make Nao respond to speech commands, avoid objects, detect faces and (3) you will combine the things you'll learn into a reactive robot.

1 Reading sensor values through ALMemory using ALModule

We will start today's lesson by walking through the Aldebaran example available at `http://doc.aldebaran.com/2-1/naoqi/sensors/altouch.html` tutorial. The NAO robot has several touch and tactile sensors; these are the colored areas on the NAO.



A: Head tactile sensor
B: Chest button
C: Tactile hands
D: Feet bumpers

Sensor values are stored in *ALMemory*. One way to read these sensor values is via the *getData()* method. A more convenient way to use sensors is to consider them as detectors of events. This holds for touch sensors (indicating whether the sensor has been touched or not), but also for collission detectors (near/far), for microphones capturing speech (word recognized or not) or cameras (face detected or not). Aldebaran has provided a convenient mechanism for its different sensors. You can *subscribe* to events on sensors by writing your own extension to the *ALModule* class of the specific module. All sensor modules can be found here: `http://doc.aldebaran.com/2-1/naoqi/sensors/index.html#naoqi-sensors`.

First we will make a class based on *ALTouch* called *ReactToTouch*, in which we specify what should be done upon detecting a particular memory change event from the touch sensors.

Code snippet 1: Example of subscribing to *TouchChanged* event

```
memory.subscribeToEvent("TouchChanged", "ReactToTouch", "onTouched")
```

For each sensor class, one or more events can be raised. The *ALTouch* generates the event *TouchChanged* to which your module can subscribe. Subscribing to an event adheres to well-known event-callback programming principles, as illustrated in the code snippet above.

Now, copy the following code, run it, and try to understand what happens. Please use a unique name where it says "yourname".

Code snippet 2: Example touch tutorial

```
# -*- encoding: UTF-8 -*-
""" Say ‘My {Body_part} is touched‘ when receiving a touch event
"""

import time
import sys
import naoqi

from naoqi import ALModule
from naoqi import ALProxy
from naoqi import ALBroker

# Global variable to store the ReactToTouch module instance
ReactToTouch = None

ip = "192.168.1.102"
port = 9559

global tts
global memory
memory = ALProxy("ALMemory", ip, port)
tts = naoqi.ALProxy("ALTextToSpeech", ip, port)

class ReactToTouch(ALModule):
    """
    A simple module able to react to touch events.
    """
    def __init__(self, name):
        #We need to unsubscribe from any proxies still on the NAO to prevent errors
        try:
            p = ALProxy(name)
            p.exit()
        except:
            pass
        ALModule.__init__(self, name)
        # No need for IP and port here because
        # we have our Python broker connected to NAOqi broker
        # Create a proxy to ALTextToSpeech for later use
        self.tts = ALProxy("ALTextToSpeech")

        # Subscribe to TouchChanged event:
        memory.subscribeToEvent("TouchChanged", name, "onTouched")
        self.footTouched = False

    def onTouched(self, strVarName, value):
        """
        This will be called each time a touch is detected.
        """
        # Unsubscribe to the event when talking,
        # to avoid repetitions
        memory.unsubscribeToEvent("TouchChanged", "ReactToTouch")

        touched_bodies = []
        for p in value:
            if p[1]:
                touched_bodies.append(p[0])
        self.say(touched_bodies)

        # Subscribe again to the event
        memory.subscribeToEvent("TouchChanged", "ReactToTouch", "onTouched")

    def say(self, bodies):
        if (bodies == []):
            return

        sentence = "My " + bodies[0]
        for b in bodies[1:]:
            sentence = sentence + " and my " + b
        if (len(bodies) > 1):
            sentence = sentence + " are"
        else:
            sentence = sentence + " is"
        sentence = sentence + " touched."

        self.tts.say(sentence)

if __name__ == "__main__":
    pythonBroker = ALBroker("pythonBroker", "0.0.0.0", 9600, ip, port)
    yourname = ReactToTouch("yourname")
    try:
        while True:
            time.sleep(1)
    except KeyboardInterrupt:
        print
        print "Interrupted by user, shutting down"
        pythonBroker.shutdown()
        sys.exit(0)
```

The callback *onTouched()* is one of the two methods that are specified in the module *ReactTo-*

Touch (the other being the initialization function). When a "*TouchChanged*" event happens, the method "*onTouched*" is called and the value of the touch event is given to this method. Subscribing and unsubscribing to events is further illustrated in the next example.

2 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 3: Speech recognition module

```
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 4: main function for module

```
if __name__ == "__main__":
    pythonBroker = ALBroker("pythonBroker", "0.0.0.0", 9600, ip, port)
    Speecher = SpeechRecognition("Speecher")
    Speecher.getSpeech(["breakfast", "pay", "checkin"], 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 touch (Section 1) 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).

3 Face Detection & Recognition

In the previous sections we learned how to write our own modules and use them for reacting on touch and speech recognition events. In this section, we are going to use a similar set up for face detection. It is important to know that there is one important difference with the previous examples: before we subscribe to the memory event, we have to subscribe to the face detection module. This lets the face detection module write to the memory, on which we can subscribe for an event. The reason for this is that if every module would write to the memory by design, the memory would soon be full.

Code snippet 5: Subscribing to face detection module

```
facep = ALProxy("ALFaceDetection", IP, PORT)
facep.subscribe("Test_Face", 500, 0.0 )
memory.subscribeToEvent("FaceDetected", "face", "faceCall")
```

The memory key we subscribe to is "FaceDetected". We define a method *faceCall()* which is called when the key's content is changed. This method has 4 arguments (self, eventName, value, subscriberIdentifier).

Rewrite the module from code snippet 5 so that it reacts to faces. Let the robot react, e.g. by saying "Hello World" when it sees a face.

The *value* in the callback method contains a lot of information about detected faces (see code snippet 6).

Code snippet 6: Structure of FaceDetected

```
FaceDetected =
[
  TimeStamp,
  [ FaceInfo[N], Time_Filtered_Reco_Info ],
    CameraPose_InTorsoFrame,
    CameraPose_InRobotFrame,
    Camera_Id
]
```

The keys in *FaceDetected()* also contain a lot information, on which you can find more information at `http://doc.aldebaran.com/2-1/naoqi/peopleperception/alfacedetection.html` This might come handy once when we are going to track and learn multiple faces.

4 Working with Sonar

In the previous sections we looked at pre-cooked examples of using touch sensors, face detection algorithms and speech recognition methods. These examples illustrate how you can easily use data and events provided by the *naoqi* framework. Now, we are going to explore working with sonar data.

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 7: 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 50cm. This involves that events associated with object detection only occur when objects are (about) in front of the NAO and closer than 50cm.

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 is part of the module and the event handler.

Code snippet 8: 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 50cm threshold actually works quite robustly…if the Nao (or approaching object) do not move too quickly.

Combining the raw sonar data (detecting objects beyond 50cm) 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.

5 Assignment

We think that you now have achieved a level of "working with the Nao" that only a few have achieved in such a short time. You know how to move the Nao (previous lesson), you have now explored face recognition, speech recognition, and touch sensors and you have some experiences with working with raw sensor data by accessing values from *ALMemory()*...

Now (actually: next lesson), show that you have really mastered these concepts by building a robot that performs well in the following context:

- Nao is standing on a black rubber mat
- the mat is enclosed by a 30cm high white wall
- the NAO can move freely and bump into the wall
- audience is standing outside the wall
- some taller objects (or human audience) are standing outside the wall

And your robot should be able to:

- react when bumping into the wall (e.g. when the toe sensors are triggered, say "Ouch" and perform a corresponding behaviour. Note that the robot may use the knowledge that objects and observers are just outside the wall) (2 points)
- avoid walls, e.g., by backing up, turning left or right and take another "walk" (2 points)
- respond to speech control commands, e.g. "stop", "move left", ... (3 points)
- When the NAO sees an object, make sure that it searches for faces and - upon seeing a face, provides some "hi there" response (3 points)