Sébastien Blin
Victor Drouin Viallard

UQAC
Université du Québec
à Chicoutimi

Université du Québec à Chicoutimi

---

# 8INF844- Projet

## Bearer Assistant

---

Sous l'encadrement de Abdenour Bouzouane

9 avril 2017

# 1 Description

TODO décrire ce qu'on souhaite mettre en place

# 2 Installation

## 2.1 Choregraphe

Choregraphe est une suite logicielle qui facilite les interactions avec NAOqi (la bibliothèque utilisée pour programmer le Nao en C++ ou Python). On peut l'utiliser pour créer des animations ou des comprotements, tester ces comportements sur des robots simulés ou réels, et obtenir un retour des composants du Nao tel que les deux caméras. On peut décrire les comportements à l'aide de boites (programmation graphique) ou en créant nos propres boites en Python. L'interaction avec NAOqi est simplifiée, mais l'exécution sera plus lente qu'un script Python ou un code C++. [Fig 1]

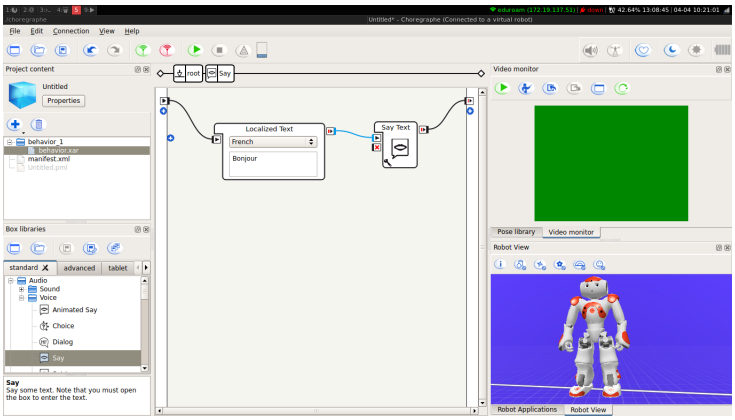


FIGURE 1 – Choregraphe

Pour l'installer il suffit de se rendre sur la page : `https://developer.softbankrobotics.com/us-en/downloads/nao-v5-v4` et de prendre le lien *Choregraphe VERSION PLATFORM Binaries* qui contiendra l'application ainsi que les bibliothèques utiles dans la section suivante.

## 2.2 Environnement Python

Nous avons choisi de ne pas utiliser Choregraphe pour la programmation du robot (pour des raisons de flexibilité, de performances) mais de le contrôler avec Python (version 2, les bibliothèques n'étant pas encore compatible Python 3). Une fois que choregaphe a bien été installé, il suffit de se rendre dans le dossier */lib* du dossier téléchargé pour obtenir les librairies utilisées en Python et C++.

Il suffit alors de copier les librairies que vous souhaitez avec votre projet afin de pouvoir utiliser NAOqi. Pour tester, vous pouvez vous rendre dans le dossier où se trouve ces bibliothèques :

```
 AmarOk@tars2 ~ : cd Downloads/choregraphe-suite-2.1.4.13-linux64/lib
 AmarOk@tars2 ~/Downloads/choregraphe-suite-2.1.4.13-linux64/lib : python
Python 2.7.13 (default, Jan 12 2017, 17:59:37)
[GCC 6.3.1 20161221 (Red Hat 6.3.1-1)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import naoqi
>>> # Ici vous pouvez utiliser NAOqi
```

## 2.3 Connexion au robot Nao

Une fois les outils installés, il est possible de se connecter au Nao afin de le controler. La première étape est d'allumer le Nao en le reliant en lien direct à votre PC ou en le connectant au WiFi (s'il est configuré pour se rendre sur un réseau WiFi).

### 2.3.1 En lien direct depuis linux

Il suffit de créer un réseau local avec le Nao. Il existe énormément de moyens de configurer ce réseau. Un des outils possible est de configurer une telle connexion est *nm-connection-editor*. Pour se faire, il suffit d'ouvrir *nm-connection-editor*, d'éditer la connexion ethernet et dans l'onglet *IPv4 Settings*, configurer le type de connexion en *Local-link only*

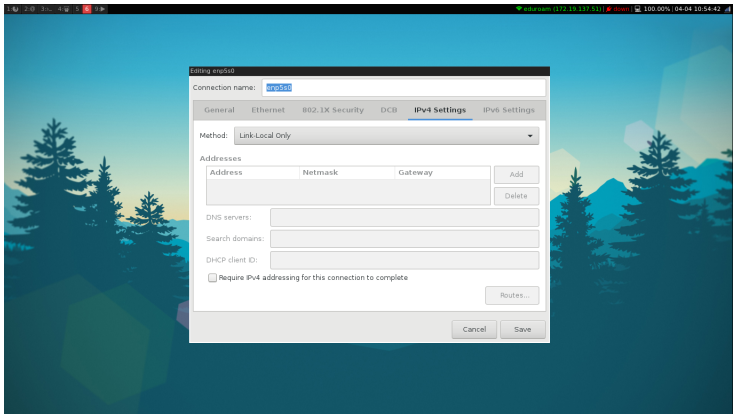


FIGURE 2 – nm-connection-editor

Après 1 ou 2 minutes, le Nao aura une adresse ip accessible, vous pourrez alors le configurer depuis l'interface du robot (*login : nao, password : nao*). Il sera par exemple possible dans l'onglet connectivité de configurer le Nao pour rejoindre un réseau WiFi.

#### 2.3.2 En lien direct depuis windows

TODO

#### 2.3.3 Via WiFi

Si le robot a été configuré pour rejoindre un réseau WiFi, il s'y connectera automatiquement et récupérera une IP.

# 3 Programmation via Python 2

## 3.1 NAOqi

### 3.1.1 Le broker

Normalement, le broker est géré de manière transparente. On en a seulement besoin si on souhaite réaliser des modules qui réagissent à un évènement via un callback. Le processus est documenté ici : `http://bx.psu.edu/~thanh/naoqi/dev/python/reacting_to_events.html#python-reacting-to-events`.

### 3.1.2 Les proxies

Un proxy se comporte comme le module qu'il décrit. Par exemple si on utilise un proxy vers *ALTextToSpeech* on aura accès aux méthodes de ce module (comme *say()* par exemple).

## 3.2 Les différents modules

### 3.2.1 Synthèse vocale [1]

Pour utiliser la synthèse vocale du Nao, il suffit de créer un proxy pour *ALTextToSpeech* :

```
from naoqi import ALProxy
tts = ALProxy('ALTextToSpeech', IP_NAO, PORT_NAO)
tts.setLanguage('French')
tts.say('bonjour')
```

### 3.2.2 Reconnaissance des visages

La reconnaissance des visages se fait à l'aide du proxy *ALFaceDetection* qui se charge d'écrire dans la mémoire s'il détecte un visage. Voici un code de base :

```
from naoqi import ALProxy
import time
```

1. `http://doc.aldebaran.com/1-14/naoqi/audio/altexttospeech-api.html#altexttospeech-api`

```
face_proxy = ALProxy('ALFaceDetection', IP_NAO, PORT_NAO)
memory = ALProxy('ALMemory', IP_NAO, PORT_NAO)
time.sleep(1)
val = self.memory.getData('FaceDetected')
print(val) # Donne les informations des visages détectés
```

La documentation de l'API est disponible ici : `http://doc.aldebaran.com/1-14/naoqi/vision/alfacedetection-api.html#alfacedetection-api`

### 3.2.3 Apprentissage et reconnaissance d'objets

Une des méthodes pour utiliser la reconnaissance d'objets du Nao est d'utiliser *Choregraphe* (la seconde est d'utiliser une application externe travaillant sur le flux de la caméra, mais ne sera pas détaillée ici). La documentation se trouve à cette adresse url : `http://doc.aldebaran.com/1-14/software/choregraphe/tutos/recognize_objects.html` et peut-être utilisé avec *Python* avec le code détaillé ici : `http://doc.aldebaran.com/1-14/naoqi/vision/alvisionrecognition-tuto.html#alvisionrecognition-tuto`

### 3.2.4 Reconnaissance vocale

Une autre possibilité que nous pouvons mettre en place est de réagir aux évènements. Par exemple, pour la reconnaissance vocale, il est possible de réaliser un module qui réagit à la reconnaissance d'un mot. Il est possible de réaliser un module de ce type (et ainsi éviter le fait de devoir écouter pendant quelques secondes et de regarder si un mot a été reconnu) à l'aide d'un broker personnalisé :

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

AgentNao = None

class Module(ALModule):
    def __init__(self, name):
        ALModule.__init__(self, name)
        self.memory = ALProxy('ALMemory')
        self.speech_recognition = ALProxy('ALSpeechRecognition')
        self.speech_recognition.setLanguage('French')

    def detect_word(self, vocabulary):
        try:
            self.memory.unsubscribeToEvent('WordRecognized', 'AgentNao')
        except:
            pass
        self.speech_recognition.setVocabulary(vocabulary, False)
```

```
        self.memory.subscribeToEvent('WordRecognized', 'AgentNao',
                                     'onWordRecognized')

    def onWordRecognized(self, key, value, message):
        self.memory.unsubscribeToEvent('WordRecognized', 'AgentNao')
        print('Recognized')
        print(key)
        print(value)
        print(message)

if __name__ == '__main__':
    broker = ALBroker('broker', '0.0.0.0', 0, NAO_IP, NAO_PORT)

    global AgentNao
    AgentNao = Module('AgentNao')
    vocabulary = ['oui', 'non', 'yes', 'no']
    words_recognized = AgentNao.detect_word(vocabulary)
    try:
        while True:
            time.sleep(1)
    except KeyboardInterrupt:
        print('Interrupted by user, shutting down')
        broker.shutdown()
        sys.exit(0)
```

La documentation de l'API de cette partie est disponible ici : `http://doc.aldebaran.com/2-1/naoqi/audio/alspeechrecognition-api.html#alspeechrecognition-api`

### 3.2.5 Postures

Ce module est utilisé pour mettre le Nao dans une position prédéfinie (comme debout, assis, etc). Les différentes postures sont disponibles ici `http://doc.aldebaran.com/2-1/family/robots/postures_robot.html` et la documentation ici `http://doc.aldebaran.com/1-14/naoqi/motion/alrobotposture-api.html`

```
from naoqi import ALProxy
posture_proxy = ALProxy('ALRobotPosture', IP_NAO, PORT_NAO)
posture_proxy.goToPosture('StandInit', 0.5)
```

### 3.2.6 Mouvement

TODO

# 4 BearerAssistant

## 4.1 Description

TODO, dire ce que fait le robot (ou ce qu'on voulait qu'il fasse)

## 4.2 Architecture du code

TODO, expliquer l'architecture du code et pourquoi c'est agent

## 4.3 Ce qui a été réalisé

TODO, fonctionnalité

## 4.4 Pistes d'amélioration

TODO (comment améliorer le Nao, ce qui n'a pas été fait)