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

18 avril 2017

1 Description

Le but de ce projet était d'utiliser le Nao pour réaliser un projet de Système Multi-agents. Après plusieurs modifications et par manque de temps pour mettre en place un projet plus grand, le Nao est utilisé pour transmettre l'objet d'une personne à une autre. Ce projet a été décomposé en deux parties. La première est de réaliser un tutoriel pour l'usage des prochains groupes. Le second est de réaliser le projet décrit plus haut avec une approche multi-agents.

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]

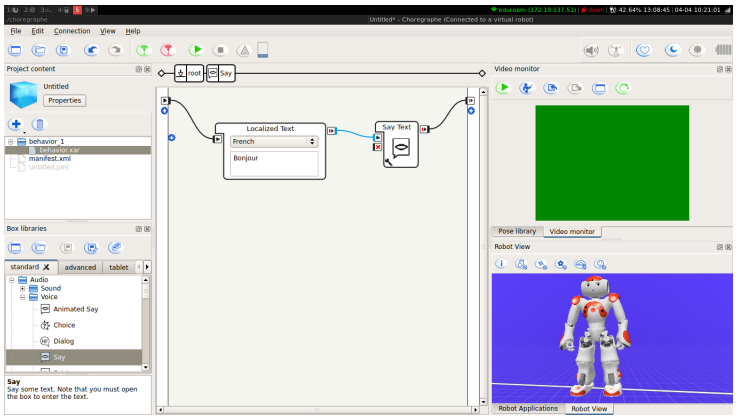

FIGURE 1 – Choregraphe

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*

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 Depuis n'importe quelle plateforme

Une seconde solution possible est de relier Nao à un routeur en ethernet et de se connecter à ce routeur par ethernet ou Wifi, ce qui permettera d'y accéder depuis l'adresse IP que le Nao aura obtenu après quelques minutes. (Il est aussi possible de le faire en lien direct, mais la procédure n'a pas été testé ici).

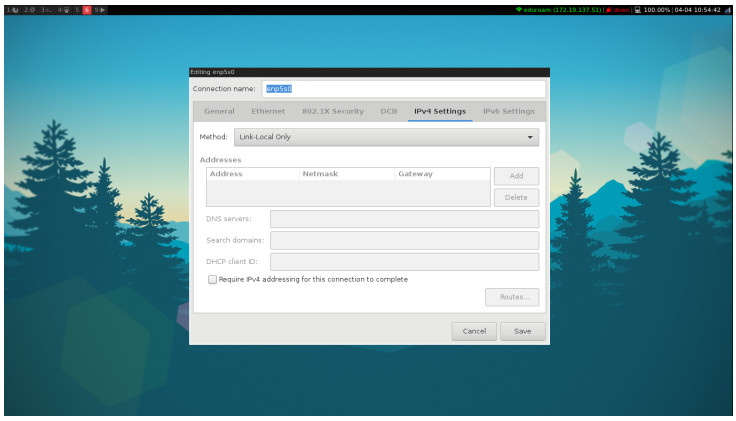

FIGURE 2 – nm-connection-editor

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* :

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

```
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

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 documenta-

tion 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

Il existe de nombreuses fonctions de faire bouger le robot. La documentation se trouve ici : http://doc.aldebaran.com/2-1/naoqi/motion/almotion-api.html

Voici quelques exempless de code pour mouvoir le Nao. Tout d'abord, pour le faire avancer :

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

motion_proxy = ALProxy('ALMotion', IP_NAO, PORT_NAO)
motion_proxy.wakeUp()
# Le faire faire demi-tour
motion_proxy.moveTo(0, 0, math.pi)
# Le faire avancer tout droit
motion_proxy.moveTo(0.2, 0, 0)
```

Il est aussi possible de contrôler les bras, par exemple faire tendre le bras au robot. Cette partie est plus complexe à prendre en main car il faut manipuler un objet Position6D. La documentation se trouve ici : http://doc.aldebaran.com/2-1/naoqi/motion/control-cartesian.html#control-cartesian

```
from naoqi import ALProxy
from naoqi import ALBroker
from naoqi import ALModule
import almath
import math
import motion

motion_proxy = ALProxy('ALMotion', IP_NAO, PORT_NAO)
motion_proxy.wakeUp()
motion_proxy.wakeUp()

effector = 'RArm'
space = motion.FRAME_ROBOT
axisMask = almath.AXIS_MASK_VEL    # just control position
isAbsolute = False
```

```
# Since we are in relative, the current position is zero
currentPos = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]

# Define the changes relative to the current position
dx = 0.1       # translation axis X (meters)
dy = 0.0       # translation axis Y (meters)
dz = 0.12       # translation axis Z (meters)
dwx = 0.00       # rotation axis X (radians)
dwy = 0.00       # rotation axis Y (radians)
dwz = 0.00       # rotation axis Z (radians)
targetPos = [dx, dy, dz, dwx, dwy, dwz]
# Go to the target
path = [currentPos, targetPos]
times = [2.0, 4.0]

motion_proxy.positionInterpolation(effector, space, path,
axisMask, times, isAbsolute)
motion_proxy.openHand('RHand')
```

Enfin, il est possible de réaliser un module réagissant à une pression sur un des multiples capteurs [2]. Par exemple :

```
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.memory.subscribeToEvent('TouchChanged', 'AgentNao', 'onTouched')

    def onTouched(self, strVarName, value):
        ''' This will be called each time a touch
        is detected.

        '''
        # Unsubscribe to the event when talking,
        # to avoid repetitions
        self.memory.unsubscribeToEvent('TouchChanged', 'AgentNao')
```

2. http://doc.aldebaran.com/2-1/family/robots/contact-sensors_robot.html#robot-contact-sensors

```
        touched_bodies = []
        for p in value:
            if p[1]:
                print(p[0])

        self.close_hand(touched_bodies)

        # Subscribe again to the event
        self.memory.subscribeToEvent('TouchChanged', 'AgentNao', 'onTouched')

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)
```

4 BearerAssistant

4.1 Description

Le sujet de notre projet a été modifié au fur et à mesure de la découverte des APIs du Nao et du temps disponible. Au début, nous souhaitions réaliser un robot assistant qui traduisait une commande en langage des signes. Au final, nous nous sommes orientés vers un robot qui se charge de demander à une personne s'il peut prendre un objet et de l'apporter à une autre personne. Au départ, il se trouve dans une position assise. Il se lève dès qu'il détecte une personne, lui demande s'il peut prendre un objet. Si la personne répond oui, le Nao tend le bras (sinon il retourne au départ) et attend un objet. Une personne lui donne alors l'objet, appuie sur le capteur à l'arrière de la main. Le nao sert sa main, baisse son bras et se retourne. Il cherche alors une nouvelle personne, va vers cette personne et lui lache l'objet. Enfin, il retourne à sa position et se rasseoit.

4.2 Ce qui a été réalisé

Au final, un prototype réalisant les fonctionnalités listées plus haut a été réalisé. Le code est réalisé sous forme d'agent réactif basé sur une machine d'états et des modules pouvant réagir aux évènements afin de réagir lorsqu'une personne touche les capteurs ou lorsque le Nao détecte une personne. Ainsi le code est décomposé comme il suit :

- *BearerAssistant* est notre Agent contenant une machine à états
- *BearerAssistantEventsHandler* est la partie récupérant les différents évènements de l'environnement pour en informer l'agent
- *states.py* contient les différents états de la machine à état. Par exemple l'état *WaitForOwner* rend le Nao en attente qu'un visage soit observé.
- *BearerAssistant* est notre Agent contenant une machine à états
- *core/agents.py* contient la définition de base d'un agent contenant une machine à état et les proxies vers les différents modules utilisés
- *core/states.py* contient la base d'une machine à état et d'un état.

4.3 Pistes d'amélioration

Tout d'abord, au niveau du code, on peut trouver quelques limites au niveau de certaines API (par exemple, la reconnaissance vocale qui marche mal ou la vision par ordinateur qui est obscure, c'est-à-dire sans retour visuel hors quelques données textuelles). Il serait donc préférable de traiter les flux sur une partie plus flexible. Une seconde piste d'améliorations serait de trouver un projet nécessitant une communication entre plusieurs robots Nao. Enfin, il serait aussi intéressant de regarder du côté des API que nous n'avons pas eu le temps d'utiliser comme l'apprentissage de nouveaux objets.