PROJET IMA4 :
COOPERATION ENTRE ROBOTS HUMANOÏDES NAO

POLYTECH'LILLE

PARTIE PROGRAMMATION DU COMPORTEMENT

Pierre Appercé | Marjorie Tixier
Promotion 2015

Marjorie Tixier
Pierre Appercé
IMA4 2013/2014

Remerciements

Avant de présenter ce rapport de projet, nous aimerions remercier les personnes qui nous ont aidés à mener à bien nos objectifs :

- M. Merzouki, pour avoir étudié et accepté notre proposition de projet pour ce semestre 8 ainsi que pour son accompagnement tout au long du projet ;

- M. Scrive, pour sa disponibilité et le prêt de nombreux outils matériels sans lesquels nous n'aurions pas pu mener à bien l'ensemble du projet.

Marjorie Tixier
Pierre Appercé
IMA4 2013/2014

Sommaire

I) Introduction générale

1) Présentation du projet

Ce projet est né de la volonté de travailler avec les NAO lors d'un projet conséquent. Le projet de coopération entre NAO consistait à faire interagir deux NAO de manière synchronisée. La finalité de ce projet était de faire porter un objet volumineux mais non lourd par les deux NAO, puis de les faire déplacer avec l'objet. Ceux-ci devaient donc repérer l'objet, s'en approcher, se synchroniser, puis porter et déplacer l'objet. Après étude du cahier des charges, nous avons remarqué que le projet pouvait être divisé en deux grandes parties : la partie comportement et appréhension de l'environnement (que nous avons réalisée) et la partie communication et réseau. C'est pour cela que nous avons souhaité travailler avec un second binôme (Céline LY et Geoffrey ROSE).

2) Cahier des charges

Dans ce projet il était question de réaliser une action de déplacement d'un objet d'un endroit à un autre de façon synchronisée.
Dans un premier temps les NAO devaient chercher indépendamment l'objet à déplacer dans la pièce, se diriger vers lui, puis attendre la synchronisation. Afin de valider la synchronisation, le réseau devait envoyer une information qui permettrait aux NAO de commencer la deuxième partie de l'action. Ils seront ainsi synchronisés pour le déplacement de la boîte d'un socle à un autre.

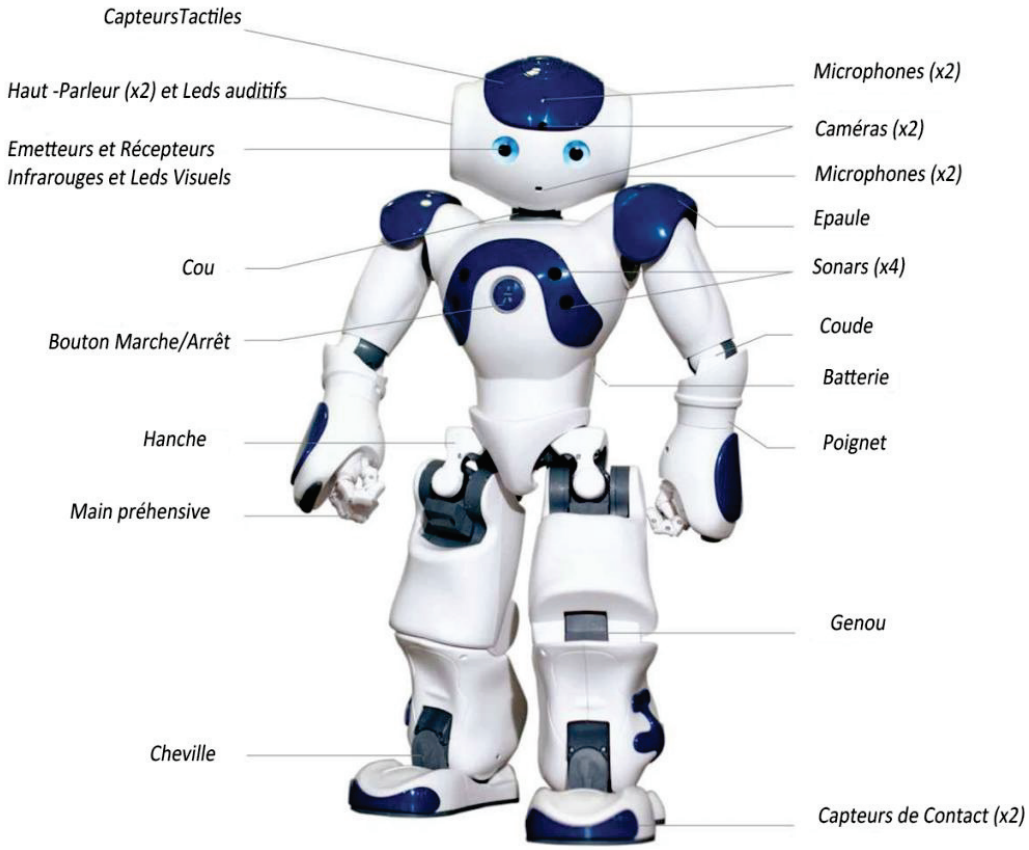
L'objectif de ce projet était donc de faire communiquer deux NAO, mais aussi d'avoir un programme facilement réutilisable et compréhensible par toute personne extérieur au projet. Bien qu'il soit possible de réaliser la programmation par différents langages, le cahier des charges nous imposait d'utiliser le logiciel fourni avec le NAO, soit Choregraphe.

II) Présentation des outils utilisés

1) Le NAO

NAO est un robot humanoïde développé en 2006 par la société Aldebaran. En 2008 cette société développe une version académique de ces robots permettant leur utilisation pour des projets ou études dans les universités et autres écoles.

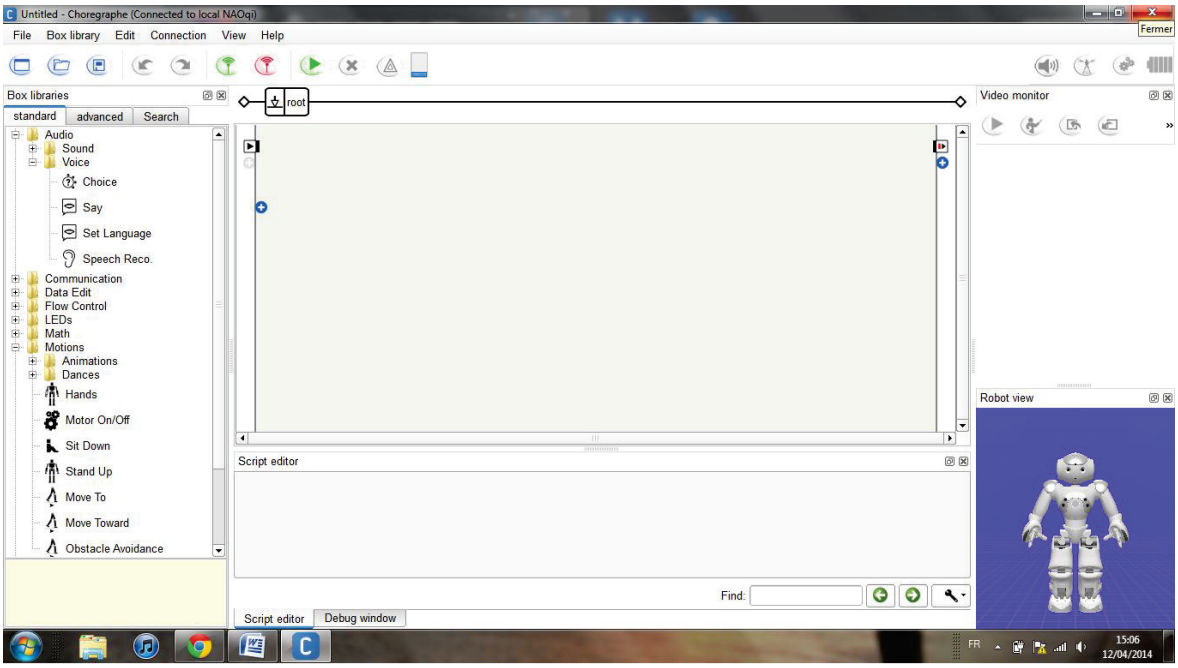
D'une taille de 58 cm et d'un poids de 5kg, ce robot possède une multitude de capteurs permettant une interaction correcte avec son environnement. Il se présente de la façon suivante :



Ce robot possède 25 degrés de liberté, ce qui lui permet de réaliser des actions proches de celles de l'humain.
Le système d'exploitation intégré au NAO est Linux, cependant le NAO peut être contrôlé à partir de trois systèmes d'exploitations : Windows, Linux et Mac OS. Les langages acceptés pour la programmation sont : C, C++, Python, Java, MATLAB, Urbi et .Net.

2) Logiciel Choregraphe

Le logiciel Choregraphe a été réalisé par la société Aldebaran dans le but de pouvoir développer graphiquement des actions pour le NAO. Il se présente comme suit :



Ce logiciel possède une bibliothèque qui regroupe plusieurs blocs permettant, par exemple, au NAO de se lever, de s'asseoir, de parler, d'enregistrer un visage….

Ce logiciel possède aussi un simulateur 3D d'un NAO. Grâce celui-ci il est possible d'observer et de faire bouger chaque articulation du NAO. Ainsi il est facile de connaitre les limites en angle de chaque articulation. D'autre part il est possible de simuler un code pour observer son comportement. Cependant, ne prenant pas en compte la gravité terrestre, ce simulateur ne permet pas de savoir si dans la réalité le NAO resterait en équilibre.

Grâce au logiciel, il est possible de créer et d'implanter un nouveau comportement dans « la tête » du NAO afin qu'il puisse l'exécuter dès sa mise sous tension. On appelle comportement un programme créé sous Choregraphe.

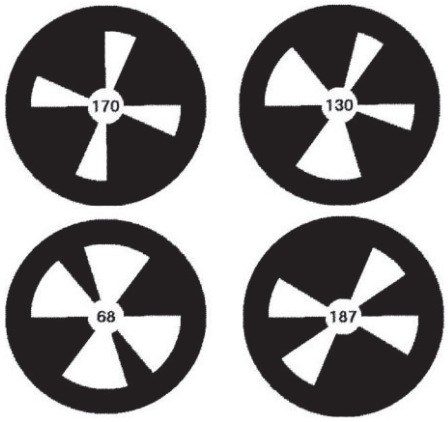
On y retrouve aussi un retour de ses caméras embarquées. La qualité présente sur le logiciel ne représente en aucun cas la véritable qualité de l'image mais permet de vérifier ce qu'observe le NAO. En effet le NAO est équipé de deux caméras HD. Mais pour que la vidéo soit fluide sur le retour caméra la qualité est grandement diminuée.

Nous avons donc utilisé ce logiciel pour développer de nouveaux comportements. Choregraphe permet aussi de réaliser ses propres blocs, sous forme de script en Python.

III) Développement du comportement

1) Déplacement des NAO vers l'objet

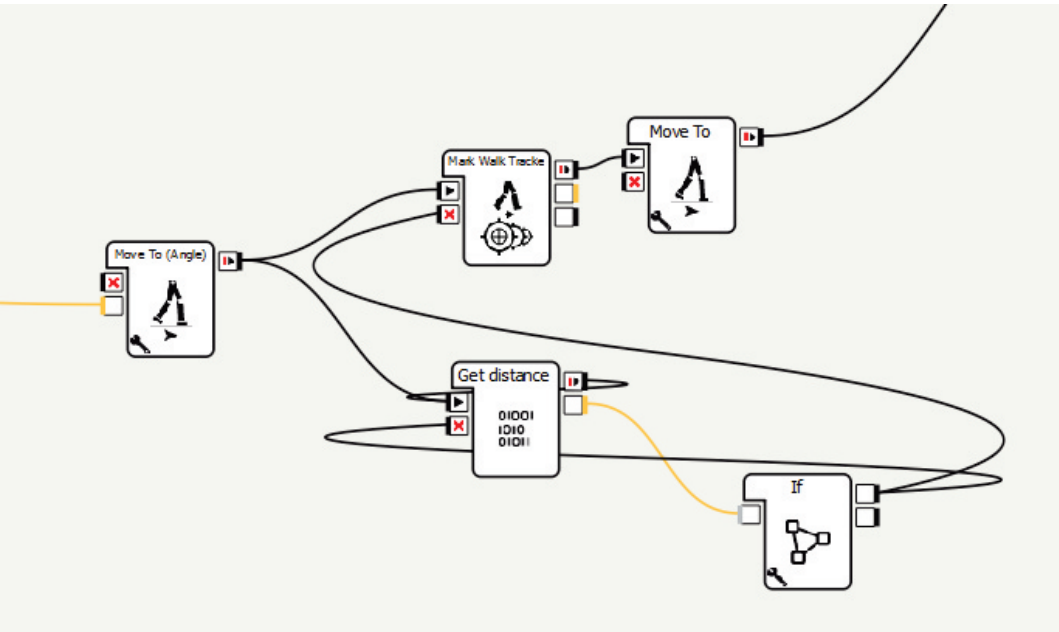
Dans un premier temps les NAO doivent repérer un objet dans une pièce, l'une des premières solutions à été de faire apprendre l'objet au NAO, cependant il lui était difficile de le reconnaître (distance, luminosité…), nous avons donc cherché une autre méthode.
Après étude de la documentation fournie par Aldebran, nous avons découvert que le NAO avait des marques préenregistrées appelées Naomarks. Il en existe plusieurs, comme suit :



Chaque Naomark possède une valeur reconnue par le NAO, cependant nous n'avons pas utilisé ces valeurs. Nous avons utilisé ces symboles comme marqueur de l'objet.

Pour que le NAO puisse repérer la boîte, il faut qu'il scrute la pièce, pour cela il effectue des pas de tête, à chaque pas on associe un angle. Le bloc « NaoMark » déjà présent dans la bibliothèque Choregraphe permet de détecter ce symbole s'il est présent dans le champ de vision du NAO. (Annexe 1)

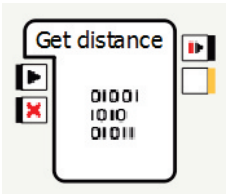
Une fois la boîte repérée il faut alors que le NAO se déplace face à celle-ci pour ensuite avancer. Pour cela nous avons utilisé le programme suivant :



Dans ce programme nous utilisons plusieurs blocs :

- Le bloc « move to angle » reçoit la valeur de l'angle de rotation que le NAO doit effectuer pour se retrouver face à la boîte et donc à la Naomark.

- Puis grâce à la fonction « Mark Walk Tracker », disponible dans la librairie de base, le NAO va s'avancer jusqu'à elle. Cette fonction est composée de plusieurs blocs.



- Après quelques essais nous nous sommes aperçu que le NAO ne s'arrêtait que lorsque la NaoMark avait totalement disparue de son champ de vision. Cependant la distance entre le NAO et l'objet était encore trop grande. Pour résoudre ce problème nous avons décidé d'utiliser les sonars. Le bloc « Get distance » nous permet alors d'activer les sonars et récupérer la distance entre le NAO et l'obstacle à intervalles de temps réguliers.

- Le bloc « If » permet d'arrêter le robot si ce dernier se trouve à quelques centimètres de la boîte. Les sonars ne pouvant calculer une distance inférieure à 20 cm, nous avons ajouté un bloc "move to" afin que le NAO se place à quelques centimètres de la boîte.

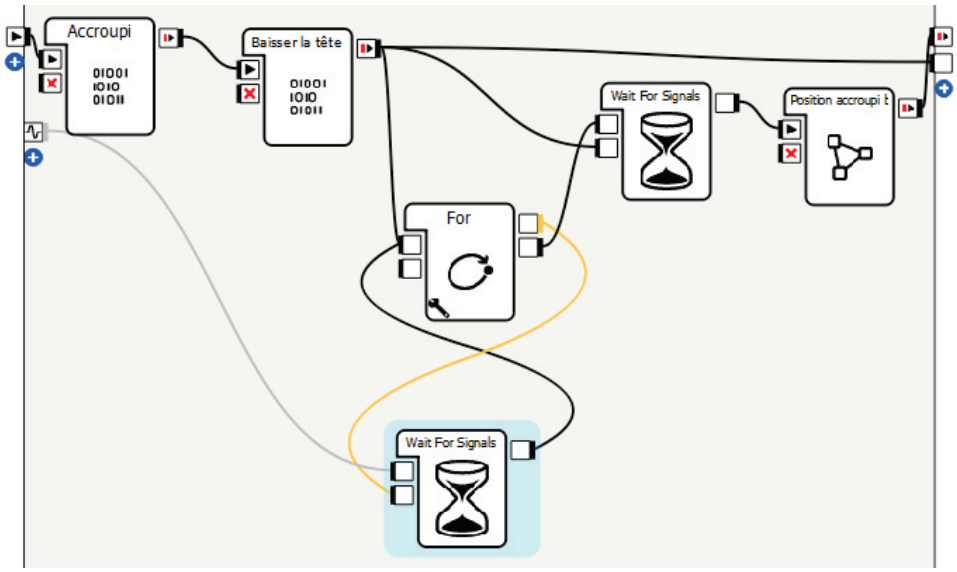
Une fois le Nao à moins de 10 centimètres de l'objet, il s'accroupit et met les bras sous la boîte. Il passe donc dans une deuxième étape qui est l'attente de synchronisation.

2) Attente de synchronisation

L'idée de ce projet était de faire communiquer via un serveur-client deux NAO pour qu'ils réalisent une action ensemble. La synchronisation est nécessaire lors de la prise de la boîte. En effet celle-ci étant grande et légère, elle risque de tomber si elle n'est pas lever des deux cotés en même temps. Il faut donc que le NAO indique qu'il est prêt à soulever la boîte et qu'il attende le second.
Nous avons décidé de ne pas définir de maître et d'esclave pour cette partie. Ainsi le premier robot qui arrive face à la boîte se signalera directement au réseau.

Voici le déroulement de la synchronisation :

Afin de signaler qu'il est prêt le NAO baisse la tête, la valeur de l'angle de l'inclinaison est récupérée par le serveur, qui peut alors faire dire au NAO "je suis prêt". Le NAO est dorénavant en position d'attente. De son côté le second NAO exécute le même programme, c'est à dire que lorsqu'il se retrouve face à la boîte il va lui aussi s'accroupir et baisser la tête afin d'indiquer au serveur qu'il est prêt. Le serveur va alors lui faire dire "moi aussi", puis pour les synchroniser l'information "allons-y" est envoyée. Les deux NAO peuvent alors poursuivre leur programme.
Voici le programme réalisé :



Le principe est de faire compter le nombre de fois qu'à parlé un NAO. Pour cela nous attendons la fin de l'évènement parler et nous incrémentons un compteur grâce au bloc « for ». Le second « Wait for signal » permet d'attendre que le for soit égal à 3 (pour « j'ai trouvé quelque chose », « je suis prêt » et « allons-y ») et que la tête soit baissée. Ainsi les deux NAO sont synchronisés et peuvent passer à la troisième étape qui est le déplacement de l'objet d'un socle à un autre.

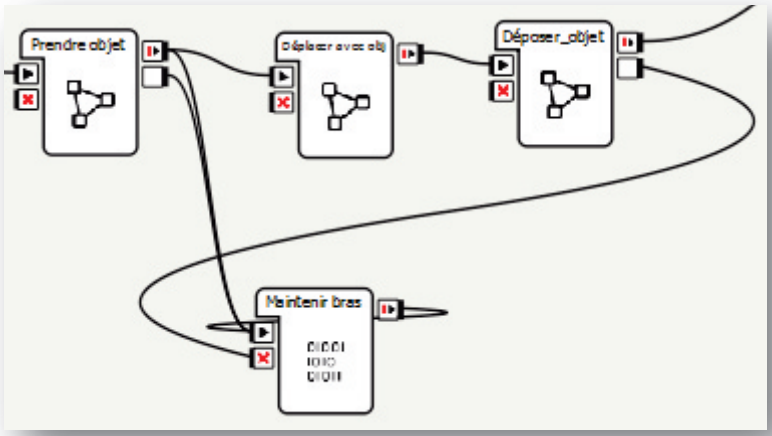
3) Déplacement de l'objet

Une fois la boîte soulevée de son socle il faut que les NAO se déplacent sans entrer en contact avec celui-ci. Pour cela nous les avons fait se déplacer sur le côté. Bien que la marche soit une fonction de base dans Choregraphe, nous avons été contraint de la modifier afin d'ajouter la position des bras désirée.

Il est important de remarquer que lorsque le NAO marche il utilise ses bras afin de se stabiliser. En effet, en fonction de la vitesse à laquelle le NAO se déplace, il commande légèrement ces bras (le bras et la jambe opposés se déplacent en même temps) de façon à garder un meilleur équilibre. Avec la position des bras que nous lui avons imposée Il ne sera donc plus capable de compenser une perte d'équilibre comme le ferait un humain.

Lorsque l'on déplace une boîte cette action avec les bras ne peut être faite. Lorsque nous avons testé notre programme nous avons rapidement vu qu'il y avait conflit au niveau des bras. En effet le NAO essayait à la fois de régler son équilibre à l'aide des bras et en même temps de les garder de manière à porter la boîte.

Nous avons donc dû dans un premier temps désactiver la commande des bras dans la fonction « walk to » et mettre en parallèle du déplacement un bloc qui permet de maintenir les bras du NAO en position.
Malheureusement en désactivant les bras de la fonction « marcher » et en les mettant en position pour porter une boîte on accroît considérablement l'instabilité du NAO lorsqu'il marche.



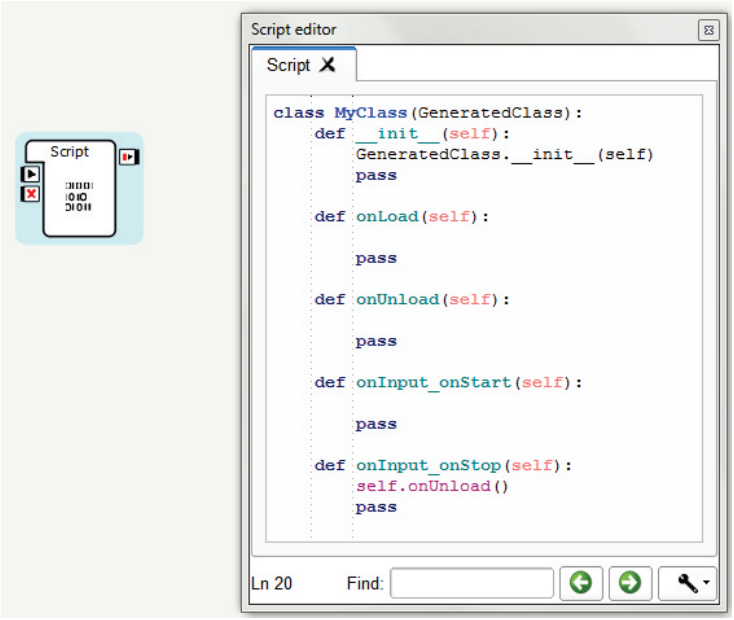
Nous pensons ainsi avoir atteint les limites du NAO lorsque celui-ci doit déplacer un objet en le portant.

4) Les scripts en Python

Nous allons dans cette quatrième partie aborder un point plus technique. L'utilisation des scripts en Python dans la réalisation de notre programme y sera développée.

Après quelques semaines de développement du comportement, les limites de la bibliothèque Choregraphe sont apparues. En effet sur certaines positions ou mouvements pré-programmés, comme la position accroupie, il nous était impossible de rajouter un mouvement de bras. D'autre part nous pouvions réaliser ces mouvements grâce à des « Timeline » (on place physiquement le NAO dans une certaine position, et l'on fait enregistrer celle-ci par Choregraphe). Cependant le programme devenait vite illisible et non modifiable.
Nous avons donc choisi de réaliser nos propres mouvements grâce au développement de scripts en Python.

Les scripts se présentent de la façon suivante :



Le bloc script est composé de 4 parties.
-La section onLoad permet de charger certains modules au lancement du bloc comme par exemple le chargement du module permettant au NAO de parler.

-La section onUnload permet de nettoyer le bloc quand celui-ci est quitté.

-La section onInput_onStart possède le code de l'action à réaliser, ainsi lorsque la boîte sera activée, soit un signal reçu sur l'entrée  , le code pourra s'exécuter.

-La section onInput_onStop permet d'arrêter la box quand l'entrée  est activée, et si besoin il y a, de renvoyer une valeur.

Il est possible de rajouter autant d'entrées et de sorties désirées.

Position accroupie :

Afin d'accroupir le NAO devant la boîte pour la lui faire prendre nous avons codé un script. Pour cela nous utilisons les six moteurs des jambes.

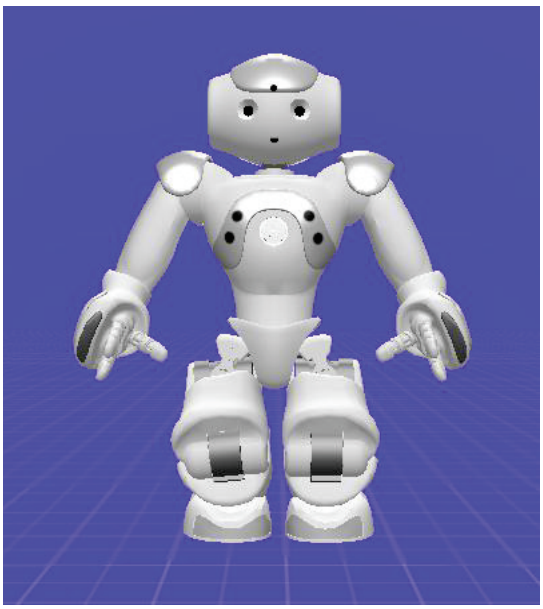
Après étude de la documentation de Choregraphe nous avons récupéré les valeurs suivantes pour le contrôle de la jambe gauche, (ces valeurs sont en radians) :

Contrôle des moteurs de la hanche:
- LHipYawPitch -1.145303 to 0.740810
- LHipRoll 0.379472 to 0.790477
- LHipPitch -1.535889 to 0.484090

Contrôle du moteur du genou :
- LKneePitch -0.092346 to 2.112528

Contrôle du moteur de la cheville :
- LAnklePitch -1.189516 to 0.922747
- LAnkleRoll -0.397880 to 0.769001

Nous les avons par la suite intégrés dans un script. Grâce au simulateur 3D du NAO nous avons pu approuver notre script, il a ensuite été possible de le tester sur le NAO réel.



Nao sur simulation sans contrôle de la tête

Exécution du script sur nao réel avec contrôle de la tête

Get angle :

Cette fonction permet de connaître la valeur des angles d'une ou plusieurs articulations. Cette fonction nous a été utile lors de la recherche des positions des bras pour porter l'objet. Afin de faciliter l'utilisation de cette fonction, nous avons développé 5 blocs (tête, jambe droite, jambe gauche, bras doit, bras gauche).

Get distance :

Nous avons eu besoin lors de ce projet de connaître la distance qui sépare le NAO d'un obstacle. Il fallait en effet que le NAO arrive jusqu'à la boîte et qu'il en soit suffisamment près pour pouvoir mettre ces bras sous la boîte mais pas trop près pour ne pas entrer en contact avec celle-ci. Nous avons donc décidé d'utiliser les sonars.

Les sonars n'étant pas activés par défaut et n'ayant pas accès à la valeur retournée par chacun des capteurs, il nous a fallu intégrer toutes ces étapes dans le script.

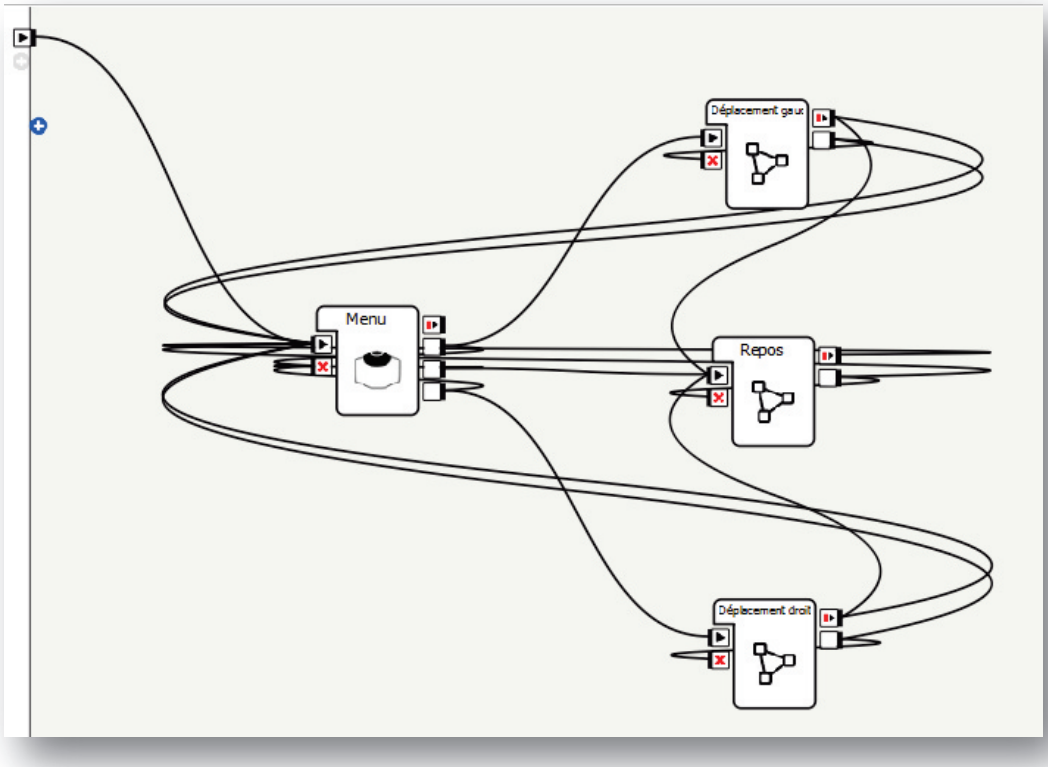
Dans un premier temps nous activons les sonars, une fois ceux-ci activés nous devons accéder à la distance séparant le NAO de l'obstacle.

Cette valeur n'est pas directement disponible, il faut dans un premier temps activer l'enregistrement périodique de cette valeur. Enfin, elle peut être utilisée en lisant dans la mémoire du NAO.

Les scripts complets de ces fonctions sont en annexe.

5) Sélection du programme

Chaque NAO est capable de réaliser le déplacement de l'objet vers la gauche ou la droite. Il faut donc pouvoir choisir dès le début le comportement qu'il va exécuter. C'est pourquoi nous avons décidé d'utiliser les capteurs tactiles situés sur le haut de sa tête.
Ainsi une fois le programme implanté dans sa tête, il va être possible de choisir soit le déplacement vers la gauche, par appui sur le capteur tactile frontal, soit le déplacement vers la droite par appui sur le capteur tactile arrière, ou bien d'arrêter le programme en le renvoyant dans sa position de repos (soit assise) par appui du capteur central.



Le programme réalisé est le suivant:
Les capteurs tactiles sont activés dès l'implantation du programme dans la tête et à chaque sortie de bloc : déplacement gauche, déplacement droite ou repos.

Le détail de la fonction « Déplacement gauche » est en Annexe 2.

IV) Conclusions

1) Problèmes rencontrés

Durant ce projet nous nous sommes heurtés à quelques problèmes. Les difficultés les plus importantes sont dues à la conception même du NAO.

- Problèmes logiciels :

Mise à jour des NAO : Deux semaines après avoir commencé le projet, nous avons constaté un défaut logiciel concernant la gestion des comportements. Il fallait remettre à jour le NAO, ceci n'a pas été chose facile, nous avons dû contacter à plusieurs reprise le service client de chez Aldebaran.

- Problèmes matériels du NAO :

Nous nous sommes aperçu que lors d'un déplacement latéral les NAO avaient tendance à partir en arc de cercle. De plus le fait de porter une boîte les déséquilibrait.
D'autre part nous voulions faire reconnaître un objet spécifique au NAO. Choregraphe possède une boîte permettant de mémoriser un objet, cependant la variation de la luminosité et des conditions extérieures ne permettaient pas une bonne détection.

2) Améliorations possibles

Au vu du résultat de notre projet trois points pourraient être améliorés.

- La recherche de NaoMark :

Dans notre programme la recherche de NaoMark fonctionne uniquement si celle-ci se trouve dans l'espace face au NAO. Si l'objet se trouve derrière le NAO alors ce dernier ne le trouvera pas. Dans ce cas, il faudrait faire tourner le NAO sur lui même pour qu'il puisse entièrement balayer la pièce.

- Le positionnement de NAO par rapport à la boîte :

Une fois que le NAO a repéré la boîte, il faudrait qu'il puisse savoir de quel côté il se trouve, et déterminer de quel côté il doit se rendre.
Nous pensons que ceci est réalisable avec l'aide des NaoMark pour connaitre les dimensions et le côté de la boîte. Afin de savoir à quel endroit se placer par rapport à la boîte, une communication entre les NAO par le biais du serveur permettrait de définir cet endroit.

- La compensation de l'erreur lors des phases de déplacement :

Lors de ce projet nous avons eu beaucoup de problèmes avec les déplacements du NAO. Lorsque le NAO se déplace dans une seule direction, sa trajectoire n'est pas droite. Le NAO se retrouve ainsi à tourner sur lui même et à se déplacer dans deux directions. Nous avons essayé de compenser cette dérivation en faisant de nombreux tests. Nous n'avons pas réussi car l'erreur de trajectoire n'est pas constante. Il faudrait coupler le NAO à un système externe (accéléromètre ou autres) pour avoir une trajectoire correcte.

3) Travail réalisé

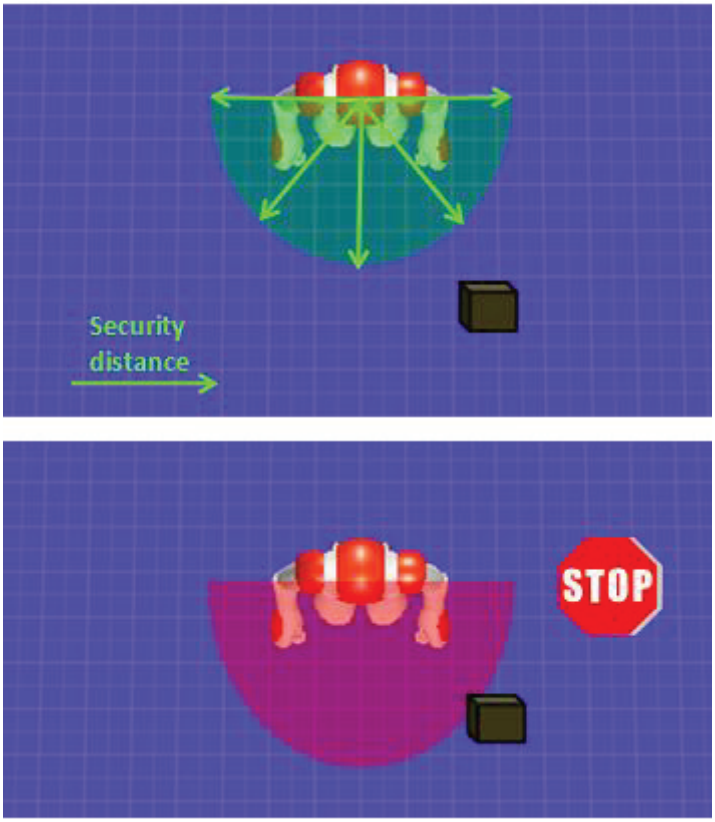
A la fin de ce projet, les objectifs fixés ont tous été réalisés dans la limite des possibilités du NAO.
En effet les deux NAO sont maintenant capable de repérer un objet grâce à une Naomark, de se diriger vers l'objet, de s'accroupir et de se relever avec lui.

La phase de déplacement a aussi été réalisée mais celle-ci n'est pas viable. Le NAO n'est pas stable durant les phases de déplacement avec un objet. Nous sommes donc arrivés aux limites du NAO dans le temps et avec les moyens dont nous disposions. Cependant nous pensons qu'avec un accès à l'ensemble des données de la centrale inertielle et une modélisation du NAO, les déplacements pourraient être grandement améliorés.

Nous souhaitions faciliter l'utilisation du NAO, c'est pourquoi nous avons fait le choix de partager notre programme sur le site d'Aldebaran Community. De plus nous avons considérablement fait évoluer la bibliothèque Choregraphe. Nous avons créé des blocs permettant la gestion des articulations des bras et de la tête, ainsi que des blocs permettant de retourner des informations sur la valeur de certains capteurs (sonar par exemple).
Nous allons transmettre cette nouvelle bibliothèque à M. Merzouki afin qu'il puisse, s'il le souhaite, la transmettre à d'autres élèves lors d'un TP ou projet.

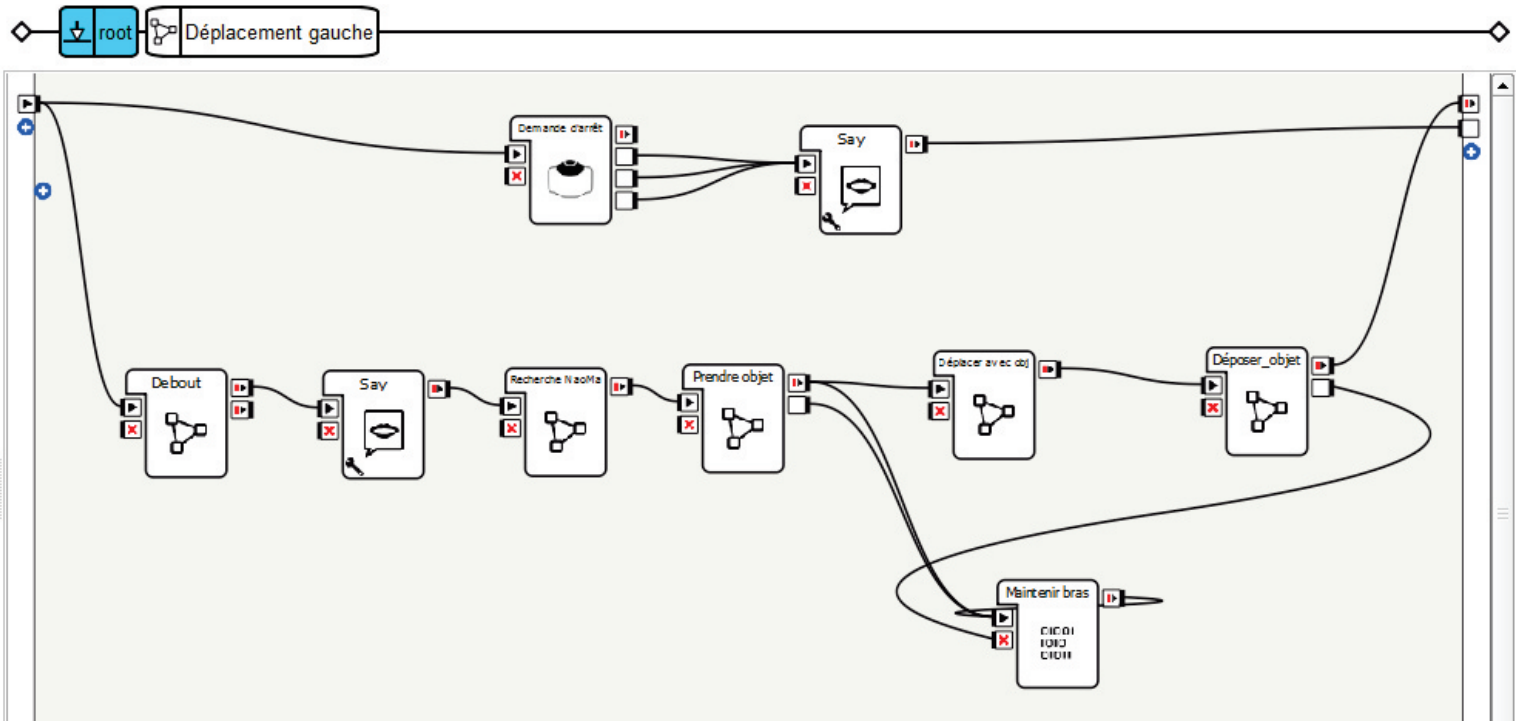
Ce projet mené en totale autonomie, nous a permis de travailler pour la première fois sur un robot humanoïde. Il nous a aussi appris à travailler dans un groupe de personnes ayant des idées différentes.
Ce fut un projet très enrichissant tant au niveau de la découverte d'un nouvel environnement de travail, que de la gestion de projet.

Annexe 1:



Annexe 2:

Déplacement gauche



```
#Position_accroupie
class MyClass(GeneratedClass):
    def __init__(self):
        GeneratedClass.__init__(self)

    def onLoad(self):
        self.tts = ALProxy("ALTextToSpeech")
        self.motion = ALProxy("ALMotion")
        pass

    def onUnload(self):
        pass

    def onInput_onStart(self):
        names = list()
        times = list()
        keys = list()


        names.append("LHipYawPitch")
        times.append([ 1.00000])
        keys.append([ 0.01538])

        names.append("LHipRoll")
        times.append([ 1.00000])
        keys.append([ -0.00456])

        names.append("LHipPitch")
        times.append([ 1.00000])
        keys.append([ -0.90962])

        names.append("LKneePitch")
        times.append([ 1.00000])
        keys.append([ 2.11255])

        names.append("LAnklePitch")
        times.append([ 1.00000])
        keys.append([ -1.18889])

        names.append("LAnkleRoll")
        times.append([ 1.00000])
        keys.append([ 0.00464])

        names.append("RHipRoll")
        times.append([ 1.00000])
        keys.append([ 0.10589])

        names.append("RHipPitch")
        times.append([ 1.00000])
        keys.append([ -0.90203])

        names.append("RKneePitch")
        times.append([ 1.00000])
        keys.append([ 2.11255])

        names.append("RAnklePitch")
        times.append([ 1.00000])
```

```
        keys.append([ -1.18630])

        names.append("RAnkleRoll")
        times.append([ 1.00000])
        keys.append([ -0.10427])

        try:
            self.motion.angleInterpolation(names, keys, times, True)
        except BaseException, err:
            pass

        self.onStopped()
        pass

    def onInput_onStop(self):
        self.onUnload()
        pass
```

```
#Get_angle
import sys
import time
from naoqi import ALProxy

class MyClass(GeneratedClass):
    def __init__(self):
        GeneratedClass.__init__(self)
        pass

    def onLoad(self):
        #motionProxy = ALProxy("ALMotion")
        self.motion = ALProxy("ALMotion")


    def onUnload(self):
        #~ puts code for box cleanup here
        pass

    def onInput_onStart(self):
        # Example that finds the difference between the command and sensed angles.
        command = 1
        names          = "Head"
        j =0

        while j<2:
            useSensors    = False
            commandAngles = self.motion.getAngles(names, useSensors)
            self.logger.info("Command angles : " + str(command))
            self.logger.info("valeur de j = " + str(j))

            useSensors  = True
            sensorAngles = self.motion.getAngles(names, useSensors)
            self.logger.info("Valeur des angles"+ str(sensorAngles))

            errors = []
            for i in range(0, len(commandAngles)):
                errors.append(commandAngles[i]-sensorAngles[i])
            time.sleep(0.5)
            j = j+1
        self.onStopped() #~ activate output of the box
        pass

    def onInput_onStop(self):
        self.onUnload() #~ it is recommended to call onUnload of this box in a onStop
        method, as the code written in onUnload is used to stop the box as well
        pass
```

```
# Get_distance
import time

class MyClass(GeneratedClass):
    global stop
    stop = 0

    def __init__(self):
        GeneratedClass.__init__(self)
        pass

    def onLoad(self):
        pass

    def onUnload(self):
        pass

    def onInput_onStart(self):
        global stop
        dataL = []
        sonarProxy = ALProxy("ALSonar")
        memoryProxy = ALProxy("ALMemory")

        #Activation des sonars
        sonarProxy.subscribe("SonarActived")
        self.logger.info("Sonar ON")
        self.logger.info(str(stop))

        #Enregistrement des données
        self.logger.info("get data")
        memoryProxy.getData("Device/SubDeviceList/US/Left/Sensor/Value")
        time.sleep(3)
        distance = memoryProxy.getData("Device/SubDeviceList/US/Left/Sensor/Value")
        self.logger.info(str(distance))

        #Envoi de la distance
        self.Distance(distance)
        self.logger.info("Distance envoyée")

        #Désactivation des sonars
        sonarProxy.unsubscribe("SonarActived")
        self.logger.info("Sonar OFF")


        if stop < 1 :
            #Activation de la sortie
            self.onStopped() #~ activate output of the box
            self.logger.info("activate output of the box")
            self.logger.info(" dans le if " + str(stop))

    def onInput_onStop(self):
        global stop
        stop = 1
        self.onUnload()
        pass
```