POLYTECH LILLE

ALDEBARAN Robotics

Rapport de projet S8

Coopération de robots humanoïdes : NAO to NAO
Partie communication / réseau

Enseignant Tuteur : M. Rochdi Merzouki

Céline Ly — IMA 4 SC
Geoffrey Rose — IMA 4 SA

Polytech'Lille 2013-2014

Université Lille1
Sciences et Technologies

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.

Table des matières

I. Introduction générale du projet de coopération de NAO

1. Présentation du projet

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

2. Cahier des charges

Nous allons tout d'abord commencer par le cahier des charges du projet global. Le projet global avait pour objectif de faire coopérer deux robots humanoïdes NAO. Ceux-ci devaient être capables d'analyser leur environnement indépendamment l'un de l'autre afin de repérer un objet de grosse taille. Ils devaient alors s'approcher, puis s'arrêter à une distance prédéterminée de l'objet. À cet instant, le premier NAO ayant atteint la cible devait attendre le second NAO, et ceci grâce à une communication réseau. La communication réseau devait permettre aux deux NAO de s'échanger leur état à l'aide d'un système serveur/client. Une fois que les deux NAO avaient pris connaissance des deux états et savaient que tous deux étaient prêts, ils devaient alors commencer à soulever l'objet. Grâce à cette synchronisation, les deux NAO pouvaient alors se déplacer avec l'objet et le déposer à un autre endroit.

3. Organisation et planning

Lors des séances de projet, nous avons décidé de séparer notre travail : l'un de nous faisait les recherches pendant que l'autre écrivait et exécutait les codes. Nous avons décidé de séparer le travail ainsi, car nous avions besoin d'un PC connecté au routeur (afin de pouvoir se connecter au NAO en *SSH*, et d'exécuter les programmes). Cette organisation s'est avérée très productive, car l'on pouvait appliquer quasiment directement ce qui avait été trouvé par le second membre du binôme. L'avantage d'une telle organisation était aussi que nous communiquions tout au long des séances, nous permettant ainsi d'être au même point dans le projet.

Au début du projet, nous avions défini trois étapes importantes :

- L'étude et la recherche de solution pour la communication de NAO vers PC et de PC vers NAO ;
- La création d'une zone de mémoire partagée ;
- La programmation de l'algorithme de synchronisation permettant aux NAO de réaliser les bonnes actions.

Les étapes peuvent être détaillées comme suit :

- Communication entre NAO et PC
  - Création d'un système serveur/client
  - Choix du protocole à utiliser
  - Tests entre deux PC, puis entre NAO et PC

- Mémoire partagée
  - Choix de la ressource ou des ressources à partager
  - Création de la zone mémoire

- Algorithme de synchronisation
  - Décision avec le second binôme du moment de synchronisation
  - Choix du type de synchronisation entre les NAO (visuellement parlant)
  - Écriture de l'algorithme
  - Tests

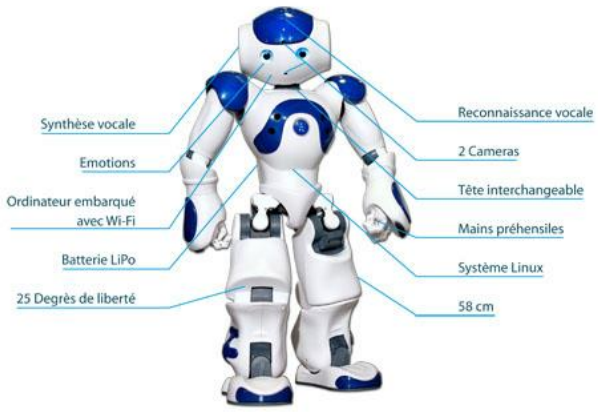
Par ailleurs, nous avions décidé que ces étapes devaient être précédées par une semaine de découverte des logiciels et outils liés au NAO.

II. Présentation des NAO

1. Le robot NAO et Aldebaran

Le NAO est un robot humanoïde autonome et programmable à 25 degrés de liberté, développé par la société française Aldebaran Robotics depuis 2006. Ce robot est capable d'effectuer de nombreux comportements, tels que marcher, parler, détecter des objets, des visages, des sons…

Il se présente comme suit :



Comme on peut le voir, le NAO est équipé de nombreux capteurs tels que :

- 2 caméras 920p permettant de traiter visuellement son environnement ;
- 4 microphones utiles pour localiser et traiter les données sonores ;
- Des capteurs capacitifs (sommet de sa tête et mains) permettant de détecter des différences de pression ;
- 2 canaux sonar afin d'estimer sa position par rapport à des objets.

Le NAO possède aussi différentes façons de communiquer. En effet, il utilise les protocoles Ethernet ainsi que Wifi afin d'être connecté à différents périphériques. De plus, il est capable d'utiliser l'infrarouge afin de communiquer avec d'autres NAO ou d'autres dispositifs travaillant avec l'infrarouge.

Concernant Aldebaran Robotics, elle est aujourd'hui considérée par nombre d'observateurs comme le *leader* mondial dans le domaine de la robotique humanoïde. En effet, le NAO équipe actuellement 480 universités ou écoles dans le cadre d'applications de recherches ou pédagogiques.

2. Les outils de travail

Le NAO peut être programmé de différentes façons :

➢ Choregraphe

Le logiciel *Choregraphe* est un outil de programmation par blocs permettant d'effectuer des actions prédéfinies par Aldebaran (marcher, avancer…). Il permet aussi de créer tous les mouvements désirés et de les faire reproduire au NAO ensuite.

➢ Langages de programmation

Le NAO peut aussi être programmé directement en langage de programmation. Il est possible de travailler en utilisant du Python, du Java, du C++ et même en programmation *Matlab*. Certaines des fonctionnalités ne sont cependant possibles qu'avec un système d'exploitation précis selon le langage choisi.

En outre, les outils de travail nécessaires pour commencer à travailler avec le NAO sont donc surtout logiciels (librairies, compilateurs…). Au niveau matériel, il sera uniquement utile de posséder un câble Ethernet et un ordinateur.

➢ Outils logiciels :

Avant de pouvoir commencer à programmer, il est nécessaire de créer un espace de travail possédant tous les outils logiciels nécessaires. Dans notre cas, nous avons choisi de programmer en C++ dans le but de parfaire nos connaissances dans ce langage.

Afin de programmer en C++, il est nécessaire d'installer Python, un autre langage de programmation orienté objet. Il servira plus tard pour utiliser des librairies spécifiques à la compilation des programmes NAO (Qibuild2).

De plus, afin de pouvoir compiler des projets C++ NAO, il est nécessaire d'installer deux outils de compilation : *CMake* et Qibuild2.

CMake est un moteur de production multiplateformes permettant de créer des fichiers de construction standards (*Makefiles*) à l'aide de fichiers de configuration appelés *CMakeLists.txt.*

En clair, c'est grâce à cet outil que nous allons lier nos différents fichiers *.cpp* et *.h* afin de créer plus tard un exécutable binaire compréhensible par le NAO.

Exemple de fichier *CMakeLists.txt* :

```
cmake_minimum_required(VERSION 2.8)
project(server)
find_package(qibuild)


set(_srcs
  server.cpp
  libthrd.h
  libthrd.cpp
  ServerSocket.h
  ServerSocket.cpp
  Socket.h
  Socket.cpp
)

qi_create_bin(server ${_srcs})

qi_use_lib(server ALCOMMON)
```

Les fichiers indiqués dans le set() sont les fichiers *.cpp* et *.h* nécessaires à la compilation du programme.

Cet outil *CMake* est ensuite appelé par un autre outil de compilation : Qibuild2. Qibuild2 est une librairie de compilation qui permet, via *CMake*, de compiler un programme C++ de différentes façons :

- Local :

Compiler en local permettra de créer un exécutable compréhensible uniquement par le PC.

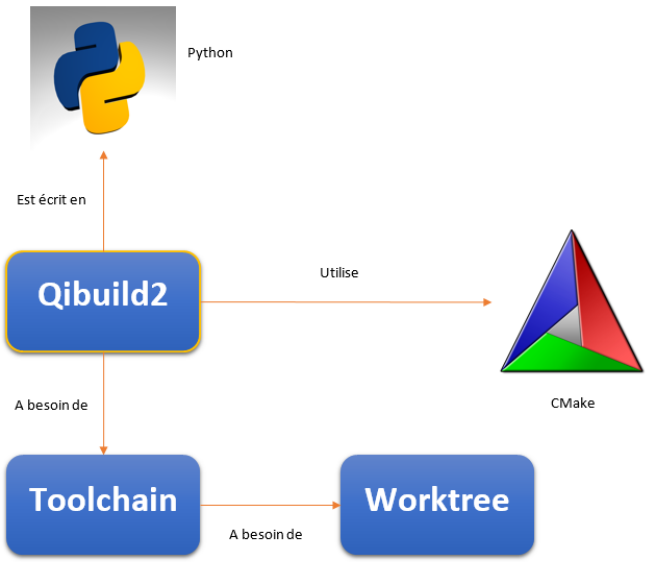
- *Geode* :

Compiler en *geode* permettra de créer un exécutable compréhensible par le NAO de version *geode*.

- *Atom* :

Compiler en *atom* permettra de créer un exécutable compréhensible par le NAO de version atom, version des NAO utilisés pour le projet.

En clair, les liens entre les outils sont les suivants :



La librairie Qibuild2 (écrite en Python), va nous permettre de déployer le fichier binaire dans le NAO afin de pouvoir l'exécuter.

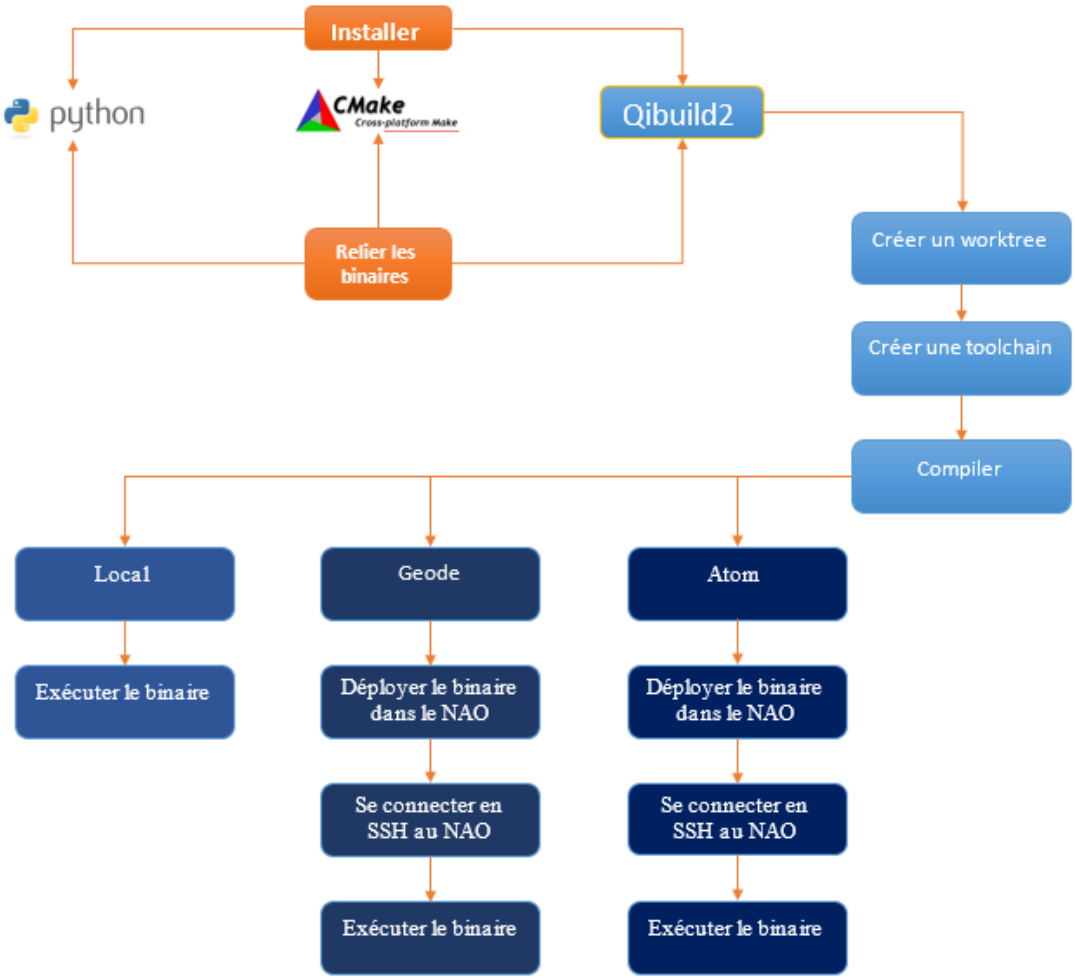
Cependant, pour que ceci fonctionne, il est nécessaire de créer un environnement de compilation à Qibuild2. Cet environnement est composé d'un *worktree* ainsi que de *toolchain*s.

Une *toolchain* est une chaîne de compilation permettant de décrire tous les paquets utilisés dans un processus de compilation de programme.

Un *worktree* va permettre de stocker toutes les *toolchain*s utilisées dans nos différents projets, une *toolchain* sera déclarée pour les projets locaux et une *toolchain* sera déclarée pour les projets à déployer en *atom*.

Une fois tout cela créé et le binaire déployé, il sera nécessaire de se connecter au NAO par le protocole de communication sécurisé *SSH*. De cette façon, on pourra ensuite accéder au fichier contenant le binaire afin de finalement le lancer de la même façon que sur Linux (*./nomdubinaire*), le NAO possédant un s*hell* Linux.

Les étapes de compilation sont alors les suivantes :



Nous avons alors pu tester ces différents outils et toutes ces étapes grâce à des programmes tests fournis par Aldebaran Robotics. Une fois la totalité des étapes effectuées et fonctionnelles, il est maintenant temps de commencer à programmer.

III. Nos réalisations

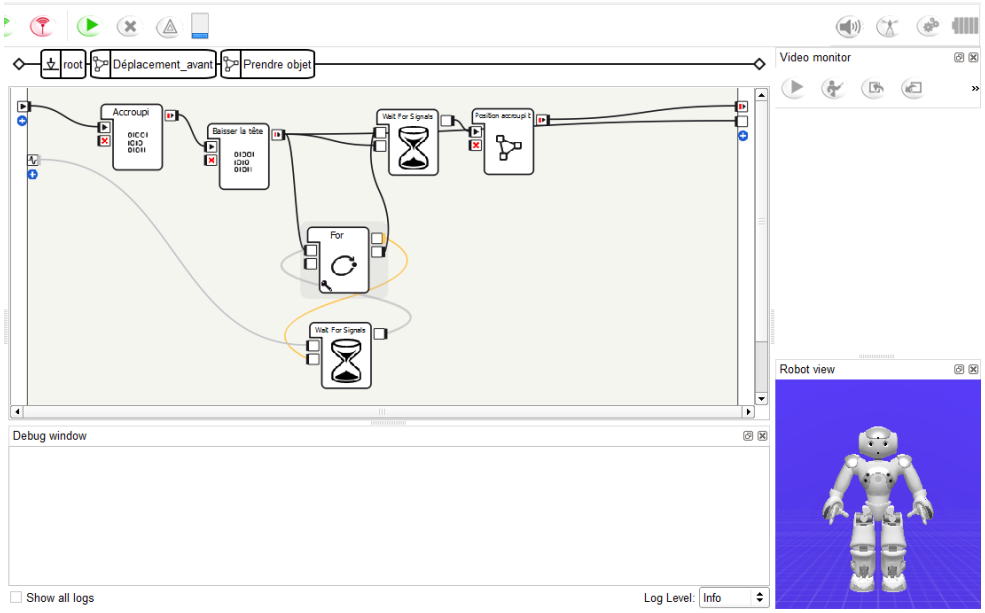
1. Algorithme de synchronisation

Notre projet étant de synchroniser les deux NAO avant la prise de la boîte, nous avons conclu avec le second binôme de la séquence suivante :

Lorsqu'un NAO est prêt, accroupi et bras tendus devant la boîte, il baissera la tête. Ce sera le signal pour indiquer qu'il est prêt à poursuivre l'action, puis il attendra que le deuxième NAO soit prêt.

Afin de créer cette synchronisation, nous avons décidé d'utiliser le module « parole » du NAO. En effet, lorsqu'un NAO sera prêt, il dira "Je suis prêt.". Lorsque le deuxième NAO sera prêt à son tour, il dira "Moi aussi.". Il faudra ensuite communiquer via un serveur pour indiquer aux deux NAO de finir cette synchronisation en disant en même temps "Allons-y!", ce qui permettra de continuer le programme *Choregraphe* créé par Marjorie Tixier et Pierre Appercé.

La synchronisation sous *Choregraphe* se présente alors de la façon suivante :



Dans cette partie du programme *Choregraphe*, on teste si le robot NAO a parlé deux fois avant de continuer le programme. La première fois correspondra à "Je suis prêt." ou bien à "Moi aussi.", tandis que la deuxième fois correspondra à "Allons-y !" (devant être dit au même moment par les deux NAO).

Dans chaque NAO, on implémentera le code suivant :

Si ma tête est baissée

Si NAO 2 n'est pas prêt et je n'ai jamais parlé :

Dire "Je suis prêt."

Sinon Si NAO 2 est prêt et je n'ai jamais parlé :

Dire "Moi aussi."

Sinon Si NAO 2 est prêt et j'ai déjà parlé :

Dire "Allons-y !"

FSi

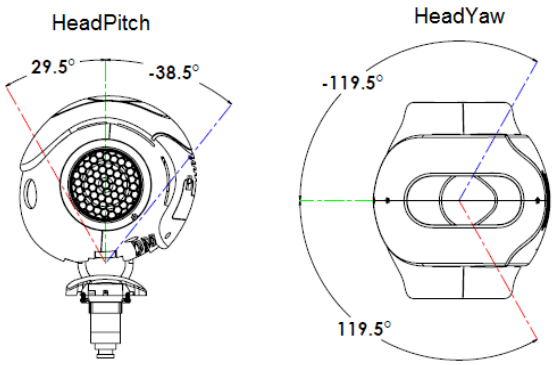
FSi

Nous devons alors réaliser plusieurs actions :

- Tester la position de la tête ;
- Faire parler le NAO ;
- Tester si le NAO a déjà parlé ;
- Tester l'état du deuxième NAO.

Tester la position de la tête

Afin de récupérer la position de la tête du NAO dans son espace, on utilise la fonction *GetData* du module *Memoryproxy* en spécifiant la clé représentant l'inclinaison de la tête (*HeadPitch*).



Après différents tests, on observe que les valeurs maximales de l'inclinaison de la tête sont 0,52 vers l'avant et -0,81 vers l'arrière.

On décide alors de tester si la tête du NAO a une valeur d'inclinaison supérieure à 0,4 et dans ce cas on considérera la tête baissée :

```
double headValue(0.4);
if((double)(memoryProxy->getData(headPitchKey)) > headValue) {…}
```

➢ Faire parler le NAO

Afin de faire parler le NAO, on utilise l'équivalent du bloc "Say" sur *Choregraphe* en C++, à savoir la fonction *TTS* (*Text To Speech*) qui s'utilise de la façon suivante :

```
tts.say(phraseToSay);
```

➢ Tester si le NAO a déjà parlé

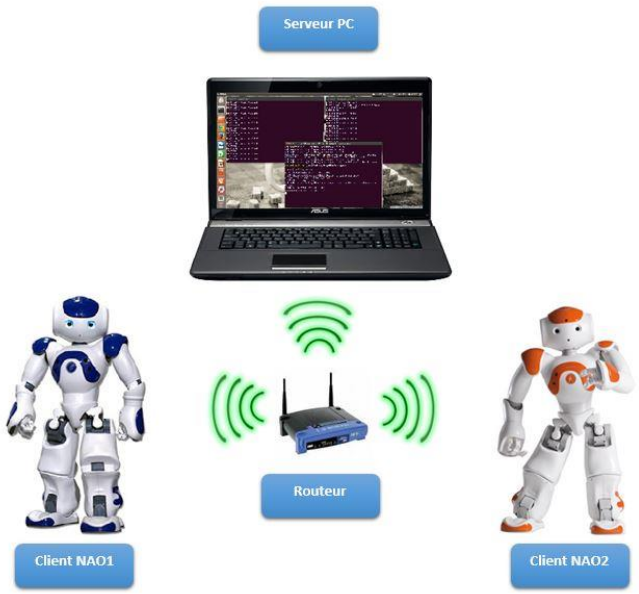
Afin de savoir si le NAO a déjà parlé, on implémente simplement une variable booléenne *talked* qui sera mise à l'état vrai lorsque le NAO aura utilisé la fonction *TTS*.

➢ Tester l'état du deuxième NAO

Afin d'avoir un retour d'informations sur l'état du deuxième NAO, on utilisera un système de serveur/client où le serveur sera en charge de communiquer l'état des NAO à chaque NAO afin de permettre au programme client (NAO) de fonctionner comme désiré.

2. Serveur/client

Comme décrit ci-dessus, le système serveur/client va permettre aux NAO de communiquer entre eux. Celui-ci a été programmé en C++, et utilise des librairies du C, car le NAO ne peut être programmé en C. Dans notre cas, un PC sous Linux fait office de serveur et les NAO sont les clients. Par ailleurs, on notera que les NAO communiquant par Wifi, il nous a été nécessaire de passer par un routeur.

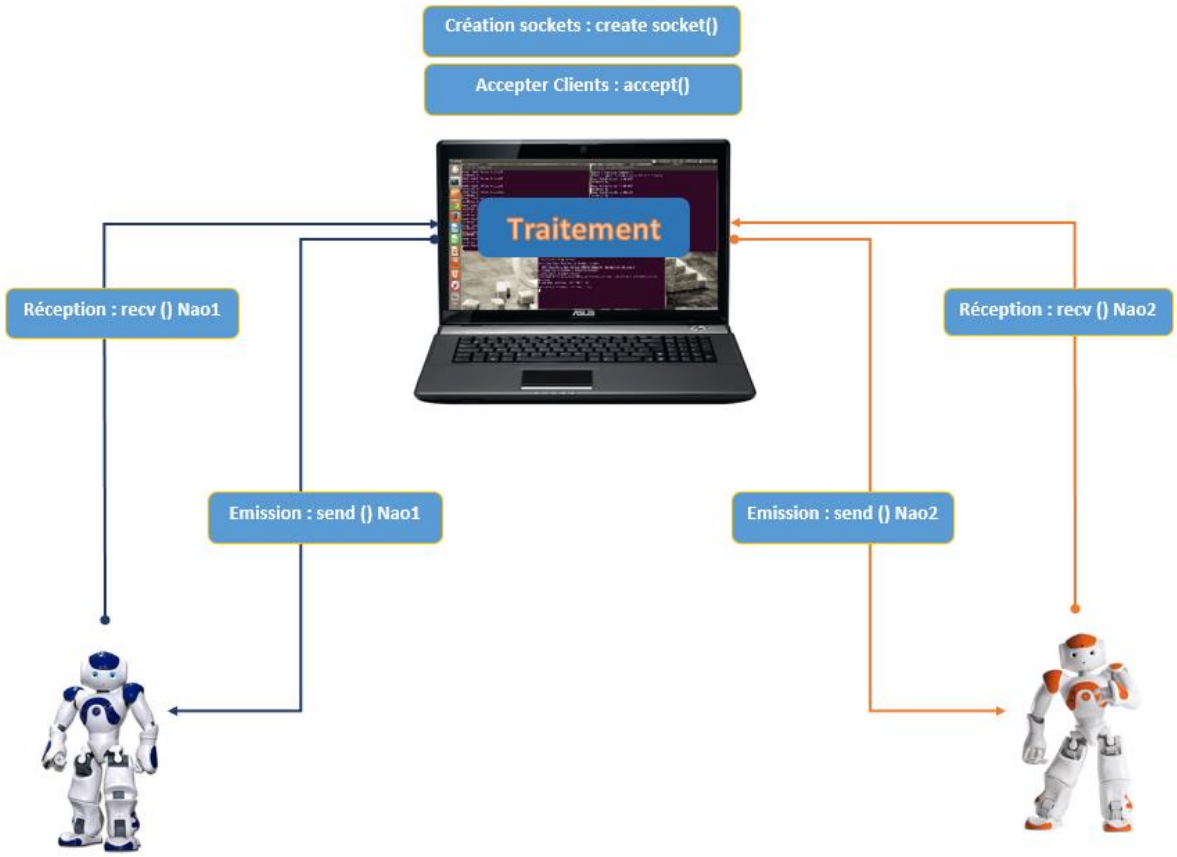


Le principe est le suivant : lorsque l'on exécute le serveur, celui-ci attend les requêtes de connexion de clients. Une fois les requêtes reçues, il accepte ces connexions. La base du système serveur/client est alors réalisée et les deux entités peuvent communiquer. Plus en détails, les étapes sont les suivantes :

- Création d'un socket, qui servira de point de communication entre le serveur et les clients : on associe alors ce socket à un port et on le paramètre ;
- Création d'un lien entre ce socket et une adresse IP afin de pouvoir retrouver ce socket ;
- Paramétrage du socket : ici, pour en faire un socket d'écoute ;
- Écoute du socket, qui va attendre que des clients se connectent ;
- Connexion des clients : la communication est effective.

Une fois cette connexion réalisée, il est alors possible d'échanger des données entre le serveur et les clients.

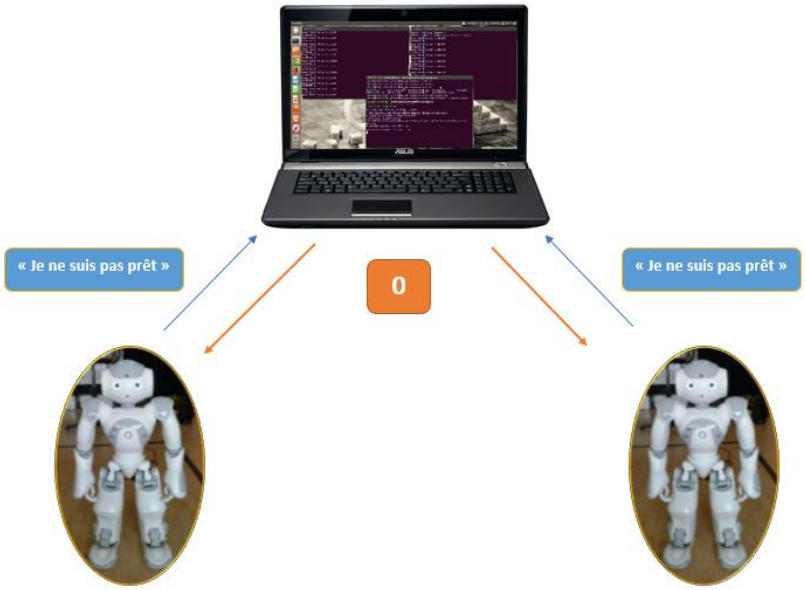
Dans notre cas, on pourra synthétiser ces étapes avec le schéma suivant :



Nous avons expliqué dans la partie précédente le côté client (NAO) de l'algorithme de synchronisation. Ici va être détaillée la partie serveur de cet algorithme; en effet, le serveur est l'endroit où les informations envoyées par les NAO sont traitées. Le serveur teste les données envoyées par les NAO, comme suit :

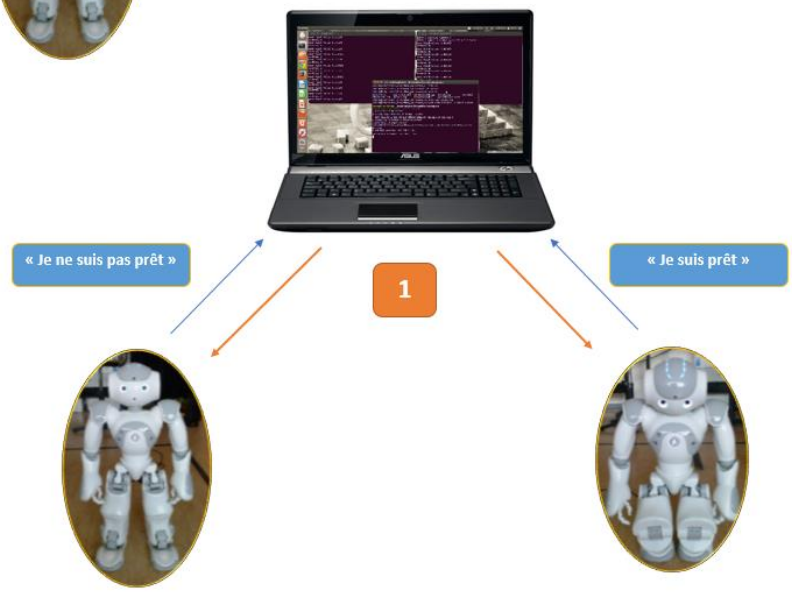
- Au début du programme, lorsqu'aucun des deux NAO n'est prêt, le serveur envoie la valeur "0";
- Si un NAO envoie "Je suis prêt.", alors le serveur renvoie aux deux NAO la valeur "1";
- Si le second NAO est prêt et envoie "Moi aussi.", alors le serveur renvoie aux NAO la valeur "2";
- Si les deux NAO sont prêts et envoient "Allons-y !", alors le serveur leur renvoie la valeur "3".

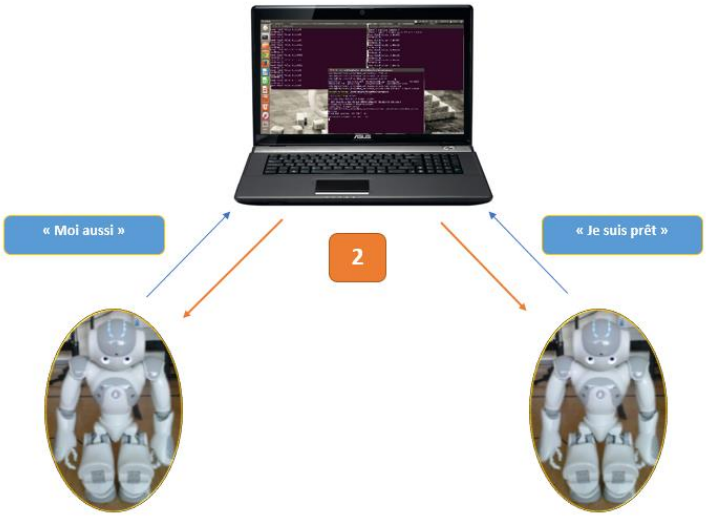
On peut alors synthétiser un déroulement normal de la façon suivante :



Etat de synchronisation 0 : aucun NAO n'est prêt, le serveur reçoit « Je ne suis pas prêt » et envoie donc « 0 » aux NAO.

Etat de synchronisation 1 : un NAO n'est prêt, le serveur reçoit « Je suis prêt » et « Je ne suis pas prêt ». Après traitement, il envoie donc « 1 » aux NAO.





Etat de synchronisation 2 : Les deux NAO sont prêts, le serveur reçoit « Je suis prêt » et « Moi aussi ». Après traitement, il envoie donc « 2 » aux NAO.

Par ailleurs, afin de ne pas revenir en arrière au niveau des étapes (renvoyer une valeur inférieure à la valeur courante), il est important que, lors de cette phase, le serveur vérifie dans quelle étape la synchronisation est située.

Cet algorithme est agencé dans le serveur de cette manière : le serveur reçoit une donnée, il la traite, avant de la renvoyer aux NAO. Ceci est réalisé jusqu'à ce que les deux NAO soient prêts. Une fois prêts, le serveur quitte le programme. À noter que du côté client, les sockets vont être fermés avant de quitter le programme, aussi.

Nous avons aussi fait le choix de réaliser ce système de serveur/client en TCP, car ceci nous permettait d'obtenir un retour d'informations; en l'occurrence, on pouvait alors savoir si les NAO avaient bien reçu les données, ainsi que leurs adresses IP afin de les distinguer.

L'avantage de ce système est ainsi de pouvoir faire communiquer plusieurs entités ensemble (ici, les NAO), en passant par un serveur, qui peut traiter les données reçues. Par ailleurs, cela nous a aussi permis de distinguer facilement les deux NAO et d'être tenus informés des éventuelles erreurs d'échange de données.

IV. Conclusion

1. Travail réalisé

Les premières semaines de projet, nous avons commencé par appréhender les logiciels, les notions et les spécificités liés au NAO. Nous avons alors principalement lu de la documentation et exécuté les exemples afin de comprendre au mieux son fonctionnement avant de le programmer.

Après cette première approche du NAO, nous avons réfléchi à ce qu'il était alors concrètement possible de faire pour synchroniser les deux NAO. Nous en avons discuté avec le second binôme du projet NAO (Marjorie Tixier et Pierre Appercé), et nous avons décidé de les synchroniser par un événement de parole. Visuellement parlant, nous avons décidé que lorsqu'un NAO était "prêt" (et attendait alors le second NAO), il devait baisser la tête.

Dans cette seconde partie du projet, après avoir apporté des précisions sur le cahier des charges, nous avons donc commencé à programmer le système serveur/client. Pour que ce système fonctionne, il fallait exécuter le serveur sur un ordinateur et les clients sur les NAO. Ceci nécessitait ainsi de réaliser une *cross-compilation*, et de se connecter via *SSH* aux NAO. Après la réalisation de ce système de serveur/client, nous avons écrit l'algorithme de synchronisation. De par la communication par TCP, les NAO ne pouvaient parfaitement être synchronisés, car l'un des deux NAO recevait nécessairement l'information après l'autre. Cependant, la synchronisation réalisée était amplement suffisante pour le reste du programme de comportement sous *Choregraphe*.

Au cours de ces dix semaines de projet, nous sommes alors parvenus à réaliser un système de serveur/client en C++ (utilisant les librairies de C pour la partie réseau), qui permet de réaliser une synchronisation de deux NAO. Ceci correspond au cahier des charges que l'on s'était fixé au début du projet.

2. Problèmes rencontrés

Au cours d'un projet, il est inévitable de faire face à des problèmes. Les problèmes que nous avons rencontrés étaient des problèmes récurrents et qui ont, malheureusement, fortement ralenti l'avancée du projet.

Le plus gros problème sans doute rencontré a été la prise en main des outils de travail, notamment des étapes de compilation. Malgré avoir lu et suivi des explications à ce sujet, il nous a fallu environ deux semaines avant de réussir à compiler un programme localement. Ce problème était récurrent, car à chaque changement d'architecture de l'environnement de travail, il fallait changer les multiples liens entre les outils de travail.

Un autre problème relativement surprenant est la version obsolète indiquée par le site d'Aldebaran de l'outil de compilation Qibuild. C'est lorsque nous avons commencé à étudier les programmes exécutables par le NAO et que nous cherchions à les déployer vers le NAO que nous nous sommes rendu compte que cette fonctionnalité n'existait pas dans Qibuild. Nous avons alors dû changer la version de Qibuild en Qibuild2, celle-ci possédant donc cette fonctionnalité indispensable à notre projet.

Un dernier problème commun aux deux groupes du projet NAO : le NAO… En effet, nous avons eu de nombreux problèmes liés directement au NAO. Le premier étant l'autonomie de sa batterie. En effet, il a besoin d'un long temps de recharge pour pouvoir être fonctionnel, mais sa batterie se décharge très rapidement… ce qui a rendu les tests plus compliqués. Les autres problèmes étaient matériels : problèmes de capteurs, problèmes au niveau des articulations, ou encore problèmes de réseau (connexion Wifi).

Malgré ces problèmes, nous avons réussi à finaliser notre projet, comme expliqué dans la partie précédente.

3. Améliorations possibles

Notre système serveur/client ne peut actuellement gérer que deux NAO. Pour améliorer cela, il aurait été bien de :

- Restructurer notre programme et utiliser un tableau (ou une liste) de structure de données : on pourrait par exemple créer une structure de données pour les NAO avec leur adresse IP et leur état;
- Utiliser des fonctions de sondage du socket (*poll()* ou *select()*) pour récupérer toutes les connexions potentielles de clients;

OU

- Utiliser des processus légers (*threads*) pour le système serveur/client, ce qui aurait amélioré l'efficacité du système et nous aurait permis de gérer facilement tous les clients potentiels.

Conclusion du projet global

Ce projet NAO regroupait deux binômes, chacun ayant un objectif précis : d'une part, travailler sur le comportement du NAO pour un environnement donné (recherche d'une cible précise et action en conséquence), d'autre part, faire communiquer deux NAO afin de réaliser une synchronisation.

Deux grands aspects peuvent être retenus lors de ce projet. Tout d'abord, nous pouvons noter l'aspect technique, avec l'application des notions de réseaux vues en cours (système serveur/client), ainsi que la découverte de nouveaux aspects de la programmation (programmation en C++, utilisation de *CMakeLists.txt* et de la *cross-compilation*). De plus, nous avons aussi appris à nous adapter à un nouvel environnement de travail, à savoir tous les outils et logiciels spécifiques au NAO. Enfin, nous pouvons aussi retenir l'aspect du travail d'équipe à deux niveaux : celui du binôme (auquel nous sommes habitués depuis quelques années) et celui du groupe de projet, qui nous a permis d'avoir un avis extérieur pertinent (de par leur propre connaissance du sujet).

En conclusion, nous pouvons alors affirmer que ce projet est une réussite tout aussi bien au niveau des objectifs que l'on s'était fixés en début de projet qu'au niveau de l'expérience acquise. Pour cela, nous tenons à remercier notre tuteur de projet, M. Merzouki ainsi que le second groupe ayant travaillé sur le projet NAO, sans qui cela n'aurait été réalisable : Marjorie Tixier et Pierre Appercé.