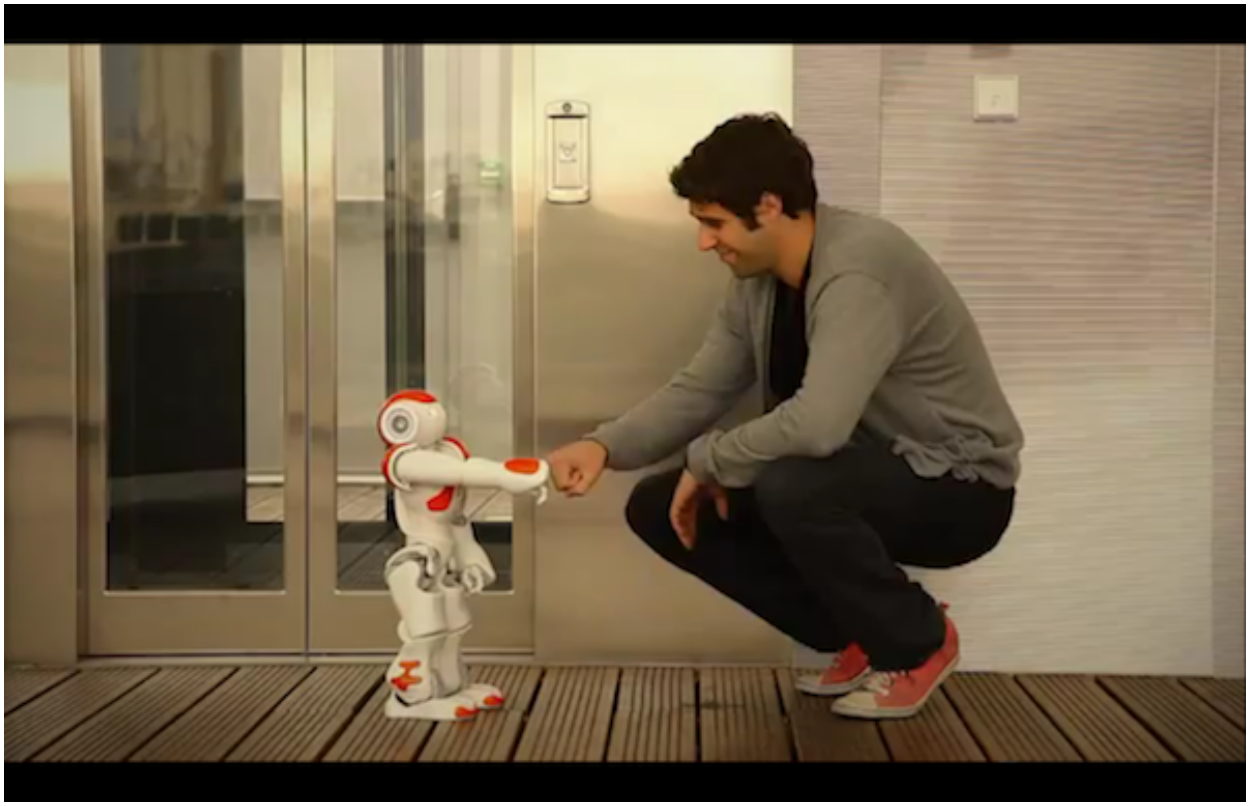
Royer Flavien

POLYTECH LILLE

Projet IMA4 : Interaction Homme - Robot



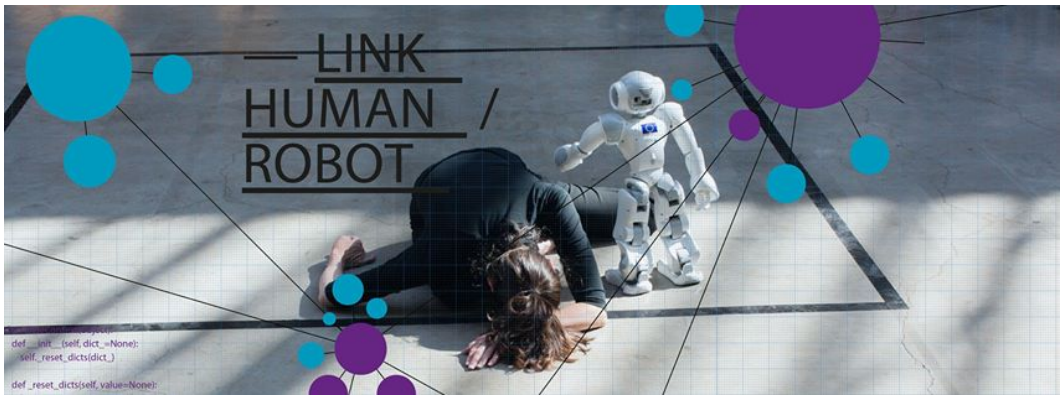
Université Lille1 Sciences et Technologies

Encadrants : Emmanuelle Grangier / Rochdi Merzouki

Sommaire

Introduction

Emmanuelle Grangier, professeur à l'école supérieure d'art de Cambrai a pour projet la réalisation d'un spectacle de danse sur le thème de la relation homme-robot nommé Link Human Robot. Lors de ce spectacle, un danseur professionnel aura comme partenaire un robot NAO avec lequel il interagira.



Notre projet IMA4 consistait à réaliser cette interaction homme robot afin de permettre à un robot NAO d'imiter les mouvements du danseur professionnel dans le cadre d'un spectacle de danse.

Pour commencer nous avons fixé en détails des objectifs du projets à l'aide d'un cahier des charges.

Cahier des charges :

Ce qui doit être réalisé :

Définition des contraintes mécaniques du robot NAO.
Acquisition des données via différents capteurs.
Traitement des données afin de faire bouger le NAO de la même manière que le danseur.
Analyse de la précision des mouvements du NAO lors de l'exécution de notre programme.

Contraintes à prendre en compte :

Notre programme doit fonctionner en temps réel et sans fil pour des raisons pratiques et esthétiques.
Les contraintes mécaniques du NAO.
Les capacités de traitement du NAO lors de l'envoi de nombreuses commandes.

Choix techniques : matériels et logiciels

Partie matérielle

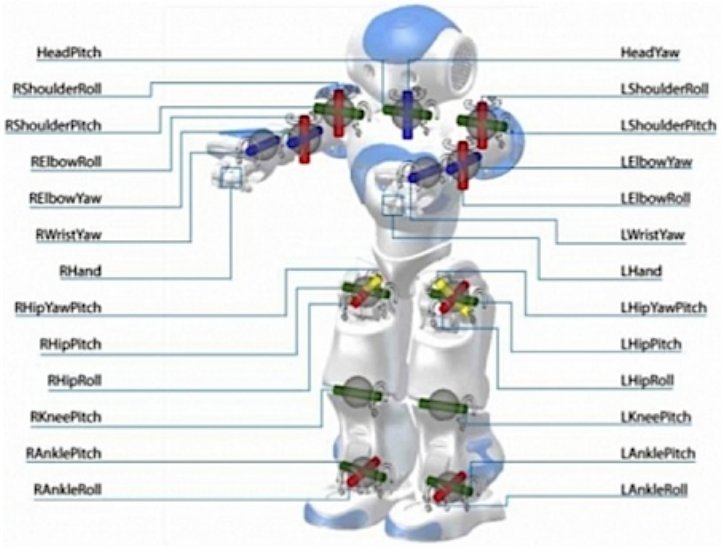
Pour réaliser ce projet, nous disposons d'un robot NAO produit par Aldebaran Robotics. Ce robot NAO comporte : une centrale inertielle située dans le torse du robot et composée de 2 gyromètres et d'un accéléromètre, des capteurs de position (des roues codeuses, situés dans chaque articulations). Il possède 26 degrés de liberté. Nous disposons aussi d'un routeur pour commander le NAO via le Wifi et d'un gyromètre afin de récupérer les données du danseur.

Partie logicielle

Afin d'interagir avec le NAO, nous utiliserons les différents logiciels disponibles sur le site d'Aldebaran:

- Chorégraphe, le logiciel permettant de programmer le NAO à l'aide de différentes fonctions se présentant sous forme de blocs.
- C++ NAOqi SDK le kit de développement qui permet de compiler un code de niveau inférieur (C++) qui communiquera directement avec l'OS du NAO.

Le langage choisi est donc le C++. Ce choix a été déterminé en fonction de la facilité de mise en oeuvre et l'efficacité de ce langage dans notre application afin de pouvoir travailler en temps réel. De plus le langage C++ ressemble fortement au code C et permet son utilisation ce qui est performant et nous est familier.



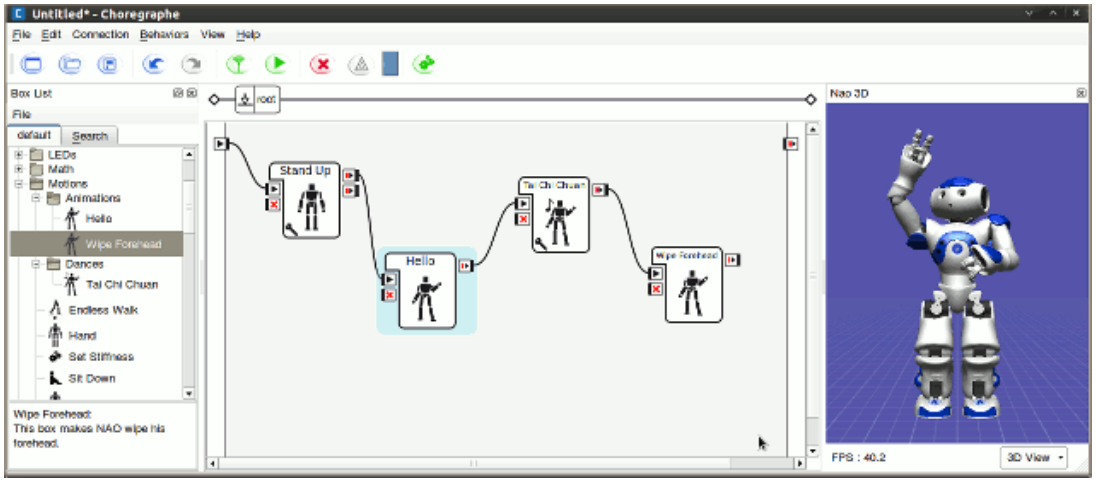
le Robot NAO et ses 26 degrés de liberté

Description du projet:

La description de ce projet sera séparée en trois parties premièrement une partie prise en main du NAO lors de laquelle nous avons découvert la librairie, installé les logiciels requis pour notre projet et réalisé nos premiers tests. Puis nous allons vous présenter la récupération des données avec le gyroscope et le code permettant au NAO de suivre l'angle défini par le gyroscope. Enfin la dernière partie présente la communication réalisée entre le NAO et le Robotino.

Première partie : Découverte du NAO et des différents softwares

Premièrement nous avons étudié le logiciel Nao Chorégraphe qui est le logiciel qui est fourni par Aldebaran Robotics afin de programmer le NAO. Il s'agit d'un logiciel de programmation par bloc assez simple d'utilisation, et il permet de réaliser un grand nombre d'actions. De plus ce logiciel offre aussi la possibilité de créer ses propre blocs en Python. Nous avons fait quelques tests avec ce logiciel comme par exemple faire danser le NAO ou bien le faire parler lorsque l'on lui touche la tête.



En revanche bien que ce logiciel offre de nombreuses possibilités dans notre cas il n'est pas possible de l'utiliser car nous devons travailler en temps réel et ce logiciel n'est pas suffisamment performant pour gérer des commandes en temps réel.

Nous avons donc choisi de coder en C++ à l'aide du SDK fourni par Aldebaran Qibuild et l'API Naoqi pour les fonctions du Nao, car le C++ est un langage très utilisé au sein d'applications temps réel.

Nous avons fait nos premiers tests avec un programme simple permettant juste de faire parler le NAO (Voir annexe 1). En revanche nous avons rencontré quelques problèmes avec l'inclusion des librairies car nous n'étions pas familier avec Cmake qui est utilisé par Qibuild.
Après quelques recherches nous avons donc réussi à inclure les librairies dans le fichier Cmakelist afin qu'elles soient bien incluse dans notre programme.

L'autre problème que nous avons rencontré était un problème de communication avec le NAO qui a été réglé en utilisant une version antérieure de Qibuild qui possède une meilleure documentation.

A partir de là nous avons testé notre programme et réussi à faire parler le NAO. Par la suite nous nous sommes donc intéressé à des fonctions plus intéressantes au sein de notre projet c'est à dire les fonctions permettant de faire bouger le NAO et plus exactement commander tous ses moteurs séparément.

Pour faire cela il y a plusieurs possibilités disponible au sein de l'API, soit réaliser un déplacement relatif de l'articulation en fonction de sa position initiale soit un déplacement absolu de celle-ci. Dans notre application un déplacement absolu semble plus intéressant étant donné que le but final est d'avoir des capteurs sur le danseur qui nous décrivent sa position actuelle.

De manière à tester cela nous avons réalisé un programme permettant un réponse à un évènement clavier. Donc notre programme consiste à faire tourner la tête du NAO à droite ou à gauche en fonction d'une entrée clavier (voir annexe 2). Cette entrée clavier sera par la suite remplacée par le traitement des données venant du capteur.

Deuxième partie : Déplacement du NAO en réponse au mouvement d'un gyromètre

Par la suite nous étions capable de faire bouger le NAO comme nous le souhaitions à l'aide d'entrée clavier la prochaine étape était donc d'acquérir les données provenant de notre gyromètre et de les traiter afin de faire bouger le NAO en conséquence.

Donc après avoir étudié la documentation technique du capteur un programme C à été réalisé afin de récupérer les données de ce gyromètre via USB. Ici nous avons rencontré un problème ou nous n'arrivions pas à récupérer des données correcte de notre capteur. Nous pensions que ce problème était lié à la configuration du port série donc nous avons cherché dans ce sens or après avoir parlé de ce problème à notre enseignant ce problème était en fait lié à la représentation des données, nous utilisions un int au lieu d'un unsigned int.

Après avoir réglé ce problème nous arrivions à traiter correctement les données provenant de notre capteur et nous avions accès à la position et l'accélération du capteur sur un axe. Nous avons incorporé ceci à notre programme précédent permettant de faire bouger la tête du NAO de manière à ce que nous puissions maintenant faire bouger sa tête de manière à suivre les déplacements du capteur.

En revanche la fonction utilisée précédemment pour faire bouger le NAO était une fonction bloquante nous avons donc eu à changer cette fonction et nous utilisons une fonction qui permet un déplacement relatif de l'articulation mais de manière non bloquante. De plus de manière à améliorer la précision de ce programme nous récupérons la position exacte de l'articulation du NAO que nous comparons à la valeur renvoyée par notre capteur afin de pouvoir ajuster la position. Ce code est présenté en annexe 3.

Pour ce programme nous obtenons une précision de l'ordre de quelques degrés (3°) et un temps de réponse de l'ordre d'une dizaine de millisecondes. La précision peux encore être améliorée en utilisant un meilleur capteur ou en améliorant le code.

Du point de vue de la robustesse quelques tests ont été réalisés avec ce même code appliqué à 6 moteurs de manière simultanée et le temps de traitement était trop long pour avoir quelque chose de fluide en revanche en ne bougeant que 2 ou 3 moteurs cela semble possible.

Les questions sur les contraintes mécaniques NAO n'ont pas été trop traitées car la documentation technique du NAO comprend toute les informations nécessaires. De plus du coté NAO il y a une sécurité en place pour que les moteurs ne forcent pas sur les articulations.

Troisième partie : Communication entre NAO et un Robotino

N’ayant pas d’exosquelette à disposition il a été décidé que pour présenter nos travaux il serait intéressant d’avoir une communication entre notre robot NAO et un Robotino. Pour cela nous avons donc utilisé un Robotino3 ainsi qu’une API permettant de programmer le Robotino en C++, robotino API2.

Robot NAO imitant une personne équipée de capteurs

Avant de commencer la communication entre ces deux robots il fut nécessaire d'étudier cette API afin de voir comment le Robotino devait être programmé. Puis nous avons réalisé un test avec un programme d’exemple qui fait décrire un cercle au Robotino. Ensuite en utilisant comme base ce programme nous avons fait un programme qui permet de déplacer le Robotino dans 4 directions (Avant, Arrière, Droite et Gauche). Ce code est disponible sur github (https://github.com/ethwynn/ProjetNao/blob/master/Move_Robotino/main.cpp) ou bien sur le wiki en annexe 4 mais n’est pas dans le rapport car assez long et donc très indigeste au sein du rapport.

Il restait ensuite à établir un moyen de communication entre ces deux programmes pour cela nous avons choisi de faire un serveur UDP qui permettrait d'envoyer les données entre les deux programmes. Le but de cette communication étant de faire déplacer le NAO suivant la même trajectoire que le Robotino les données devant être envoyées seront donc 3 float pour la position en x, en y ainsi que l'angle de lacet phi. (Ce code n'est pas fourni en annexe mais il est disponible ici https://github.com/ethwynn/Server-UDP)

Au début le serveur UDP réalisé permettait de transmettre un structure comportant les 3 floats en question en revanche au moment de l'implantation sur le Robotino cela n'était plus fonctionnel. En raison du manque de temps nous avons donc décidé d'envoyer nos 3 floats séparément suivis d'un dernier float de séparation. De cette manière nous avons réussi à communiquer entre nos deux programmes.

Ensuite il restait à faire déplacer le NAO suivant la même trajectoire que le Robotino donc du coté Robotino il fallait récupérer les données concernant sa position via la fonction d'odométrie. Ensuite les envoyer, les traiter et déplacer le NAO en conséquence. Il fut nécessaire de faire un léger traitement des données car les positions envoyées par le Robotino sont des positions absolues alors que pour le déplacement du NAO il nous fallait des déplacements relatifs. Ainsi qu'un léger ajustement des distances qui sont différentes entre le NAO et le Robotino.

Nao et Robotinos

Au final cela fonctionne on arrive à faire déplacer le NAO suivant les déplacements du Robotino, il était prévu au départ de faire la communication dans les deux sens c'est à dire d'implémenter en plus des actions sur le Robotino dans le cas ou l'on actionne certains capteur du NAO mais faute de temps cela n'a pas été réalisé.

Conclusion

Pour conclure le cahier des charges a été rempli bien que l'on aurait souhaité pouvoir en faire plus. On peux déplacer le NAO en fonction des données renvoyées par un ou plusieurs capteurs et cela avec une bonne précision et en temps réel. Nous n'avons pas généralisé le code pour avoir du code fonctionnel avec les 26 moteurs du NAO mais il est juste nécessaire de changer le nom du joint à bouger pour pouvoir utiliser autre chose que la tête.

Le projet n'est donc pas tout à fait utilisable tel quel il manquerait quelques finitions dont une interface graphique qui n'a pas été réalisée.

Pour ma part ce projet fut très intéressant car j'ai eu l'occasion de travailler sur le robot NAO sur lequel je n'avais jamais travailler avant et qui est un des symboles de l'IMA. Ce projet m'a aussi permis de travailler en autonomie car nous savions ce que nous avions à faire mais nous avons eu à faire des choix par nous même pour les méthodes utilisées. Ce robot est en plus produit par Aldebaran Robotics qui est une entreprise au sein de laquelle j'aimerai travailler donc ce projet est pour moi une très bonne expérience.

Annexes

Annexe 1

```
#include <iostream>
#include <stdlib.h>
//Librairie permettant la création du proxy
#include <alcommon/alproxy.h>

int main(int argc, char* argv[])
{
  // Nous allons connecter notre Broker a Naoqi
  int pport = 9559;
  std::string pip = "127.0.0.1";

  // On teste le nombre d'arguments
  if (argc != 1 && argc != 3 && argc != 5)
  {
    std::cerr << "Wrong number of arguments!" << std::endl;
    std::cerr << "Usage: mymodule [--pip robot_ip] [--pport port]" << std::endl;
    exit(2);
  }

  // Si il y a un seul argument il s'agit de l'IP ou du PORT
  if (argc == 3)
  {
    if (std::string(argv[1]) == "--pip")
      pip = argv[2];
    else if (std::string(argv[1]) == "--pport")
      pport = atoi(argv[2]);
    else
    {
      std::cerr << "Wrong number of arguments!" << std::endl;
      std::cerr << "Usage: mymodule [--pip robot_ip] [--pport port]" << std::endl;
      exit(2);
    }
  }
```

```
// Si il y a deux arguments on spécifie le PORT et L'IP
 if (argc == 5)
 {
   if (std::string(argv[1]) == "--pport"
       && std::string(argv[3]) == "--pip")
   {
     pport = atoi(argv[2]);
     pip = argv[4];
   }
   else if (std::string(argv[3]) == "--pport"
            && std::string(argv[1]) == "--pip")
   {
     pport = atoi(argv[4]);
     pip = argv[2];
   }
   else
   {
     std::cerr << "Wrong number of arguments!" << std::endl;
     std::cerr << "Usage: mymodule [--pip robot_ip] [--pport port]" << std::endl;
     exit(2);
   }
 }

 /**
  * On crée un proxy qui permet d'appeler un module défini dans l'API
  * La méthode pour appeler ce module ce présente comme ceci :
  * AL::ALProxy proxy(<module_name>, <robot_IP>, <robot_port>);
  */
 // Création d'un proxy vers ALTextToSpeechProxy
 AL::ALProxy proxy("ALTextToSpeech", pip, pport);

 /**
  * Si la methode ne renvoie rien
  * On l'appelle de cette manière :
  * proxy.callVoid(<bind_method>, <parameter>, ...)
  */
 // On appelle la méthode Say
 proxy.callVoid("say", std::string("Bonjour je m'appelle NAO!"));

 return 0;
}
```

Annexe 2

```
#include <iostream>
#include <stdlib.h>
#include <stdio.h>

#include <alcommon/alproxy.h>          //Librairie permettant la création du proxy
#include <alproxies/almotionproxy.h>    //Librairie contenant les fonctions pour faire bouger le NAO

int main(int argc, char* argv[])
{
  // Nous allons connecter notre Broker a Naoqi
  ...
  Toute la partie connexion est identique au programme précédent.

  /** Le nom de la partie que l'on souhaite bouger */
  const AL::ALValue jointName = "HeadYaw";      //Ici il s'agit de la tête

  try {
    /** On cree notre proxy qui servira a appeler les methodes pour bouger la tete
    * Les arguments du constructeur sont :
    * - L'adresse IP du NAO
    * - Le port utilisé. Par défaut il s'agit du port 9559
    */
    AL::ALMotionProxy motion(pip, 9559);


    /** Afin de bouger la tête on doit vérifier qu'elle n'est plus asservie
    * Pour cela on va mettre la "dureté" de la tête a la valeur maxi, c'est dans cette position que l'on
peut bouger la tête
    */
    /** Dureté voulue */
    AL::ALValue stiffness = 1.0f;
    /** Temps en seconde pour atteindre cette valeur */
    AL::ALValue time = 1.0f;
    /** Appel de la méthode pour réaliser l'action */
    motion.stiffnessInterpolation(jointName, stiffness, time);


    /** Ensuite on va bouger la tête
    * Pour cela on va lui donner des angles et le temps utilisé pour atteindre ces angles
    */
```

```
    /** On défini une liste d'angles, en radians. */
    AL::ALValue targetAngles = AL::ALValue::array(-1.5f, 1.5f, 0.0f);
    /** On defini les temps correspondants, en secondes. */
    AL::ALValue targetTimes = AL::ALValue::array(3.0f, 6.0f, 9.0f);
    /** Ensuite on peut choisir si les angles sont des positions absolues ou relatives par rapport a la
position actuelle de la tête. */
    /** Ici se sont des angles absolus. */
    bool isAbsolute = true;

    /** Puis on appelle notre fonction. */
    motion.angleInterpolation(jointName, targetAngles, targetTimes, isAbsolute);


    /** Pour finir on remet la dureté a sa valeur initiale. */
    stiffness = 0.0f;
    time = 1.0f;
    motion.stiffnessInterpolation(jointName, stiffness, time);


    }
    catch (const AL::ALError& e) {
      std::cerr << "Caught exception: " << e.what() << std::endl;
      exit(1);
    }
exit(0);
}
```

Annexe 3

```
//Gestion du SIGINT afin de permettre l'arrêt du programme
void hand(int sig){
        if (sig == SIGINT){
                printf("SIGINT\n");
                inter = false;
        }
        else{
                printf("bizarre !\n");
        }
}

int main(int argc, char* argv[])
{
  //Connexion
  ...

  /** Nom du joint à bouger. */
  const AL::ALValue names = "HeadYaw";
  int fd;
  double angle;
  unsigned char buf[8] = {0};
  action.sa_handler = hand;
  sigaction(SIGINT, &action, NULL);

  init_serial(&fd); /Initialisation du port serie

  AL::ALMotionProxy motion(pip, 9559);
  AL::ALRobotPostureProxy robotPosture(pip,9559);

  try {

    //Make the NAO stand
    robotPosture.goToPosture("Stand", 1.0f);

    AL::ALValue stiffness = 1.0f;
    AL::ALValue time = 1.0f;
    motion.stiffnessInterpolation("Head", stiffness, time);

    double newAngle=0.0,a0=0,rangle;
```

```
    while(a0 == 0){
        getData(buf,fd);
        a0 = getAngle(buf);
    }

    while(inter){
        getData(buf,fd);
        angle = getAngle(buf);
        if (angle != 0){
            rangle = angle -a0;
        }
        if (rangle > 180){
            rangle = rangle - 360;
        }
        else if (rangle < -180){
            rangle = rangle + 360;
        }
        rangle = (rangle*PI)/180;
        std::vector<float> sensorAngle = motion.getAngles(names, true);

        double changes = rangle - sensorAngle[0];
        std::cout << "changes: " << changes << std::endl;
        float fractionMaxSpeed = 0.10f;
        // Appelle la fonction changeAngles qui permet de bouger un joint à la vitesse désirée
        if(changes > 0.07 || changes < -0.07){
            motion.changeAngles(names, changes, fractionMaxSpeed);
        }
        else{
        }
    }
    stiffness = 0.0f;
    time = 1.0f;
    motion.stiffnessInterpolation("Head", stiffness, time);
    robotPosture.goToPosture("Sit", 1.0f);
    }
    catch (const AL::ALError& e) {
      std::cerr << "Caught exception: " << e.what() << std::endl;

      robotPosture.goToPosture("Sit", 1.0f);
      exit(1);
    }
exit(0);
}
```