MONIK

IMM – MASTER // HES-SO

MMI Operation : NAO Interaction with Kinect (Monik) est un projet visant à explorer les possibilités de pilotage d'un robot NAO par le mimétisme

Baudin Sébastien, Maillard Martin, Tscherrig Julien

10/06/2011

Table des matières

1 Introduction

MMI Operation : NAO Interaction with Kinect (Monik) est un projet visant à explorer les possibilités de pilotage d'un robot *NAO* par le mimétisme. Ce genre d'interaction est déjà utilisé dans plusieurs domaines, comme la chirurgie de précision où un spécialiste peut opérer un patient à distance par le biais d'un robot imitant ses gestes. Nous voulons appliquer ce principe au robot *NAO* en utilisant le dispositif *Kinect* de *Microsoft* pour capturer les gestes de l'utilisateur, les commandes vocales permettront d'effectuer des opérations spécifiques décrites plus tard dans ce document et le feed-back du robot – à savoir les images fournies par sa caméra frontale – sera projeté dans des lunettes afin d'avoir une immersion maximale. Le choix du mimétisme est venu spontanément parce qu'il nous paraît être la solution la plus naturelle de commander un robot à distance. Plusieurs problématiques sont à envisager, comme la place nécessaire aux déplacements ou la précision que l'on pourra obtenir.

NAO est un robot humanoïde crée par *Aldebaran Robotics* et destiné au milieu de recherche académique. Il est fourni avec un environnement de programmation qui se veut simple d'utilisation.

Kinect est un périphérique initialement destiné à la console de jeux vidéo *Xbox 360*. Il s'agit en fait d'une caméra 3D et plusieurs librairies permettant la détection du squelette des utilisateurs (la position de leurs membres) sont disponibles. Son utilisation est aujourd'hui largement répandue dans le milieu de la recherche.

Ce projet ne constituera que les prémices d'une nouvelle forme d'interaction à distance mais dans un hypothétique développement idéal de la solution, plusieurs scénarii d'utilisation sont imaginables. Par exemple, certains environnements dangereux sont difficilement accessibles à des êtres humains, notamment les zones irradiées ou les décombres instables de bâtiments, et nécessitent néanmoins d'y effectuer certains travaux. Un appareil motorisé humanoïde pouvant être commandé naturellement à distance par un humain rendrait ces opérations beaucoup plus simples. Il faudrait évidemment un système très précis et un robot protégé contre les dangers potentiels.

2 Architecture du système

Comme présenté sur la Figure 1 - Architecture, les entrées du système sont la *Kinect*, un microphone (headset) et les accéléromètres des lunettes de réalité augmentée. Les événements générés par ses dispositifs sont traités par notre moteur de fusion, qui va ensuite indiquer au robot les mouvements à effectuer. Le feedback à l'utilisateur est l'affichage de la vue de la caméra du robot (ses « yeux ») dans les lunettes de réalité augmentée de l'utilisateur.

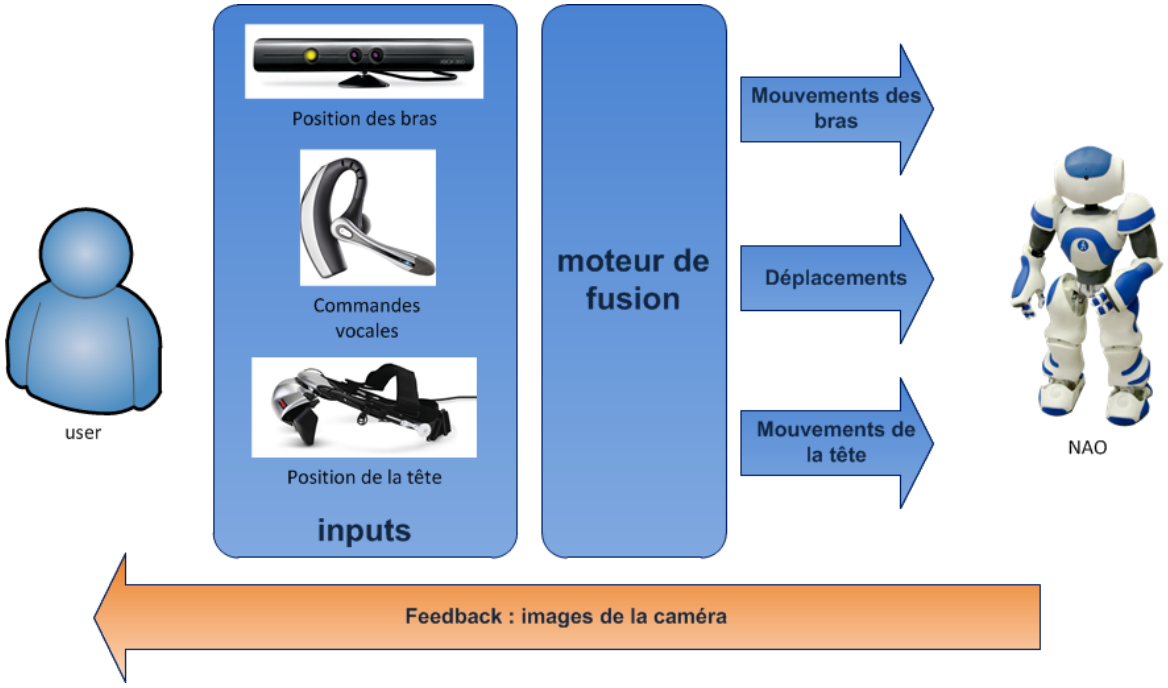


Figure 1 - Architecture

2.1 Caméra 3D – Kinect

La *Kinect*, une caméra 3D développée par *Microsoft*, est employée pour la détection des mouvements d'une personne. Elle a été créée pour être employé sur la console *XBOX 360*, mais plusieurs kits de développement existent pour qu'elle puisse être employée sur des ordinateurs.

2.2 Robot – NAO

Nao est un robot humanoïde, programmable et mesurant 58cm. Il a été développé par *Aldebaran Robotics*, une société française. Plusieurs logiciels, ainsi que des kits de développement, sont développées par cette même société afin de manipuler le plus facilement possible ce robot.

Il dispose de 25 axes qui peuvent être réglés indépendamment, ce qui permet un grand degré de manipulation. Deux caméras sont disponibles, une regardant à l'avant et l'autres regardant ses pieds. Plusieurs capteurs, de pressions et infrarouge, permettent de détecter les objets devant le robot.

2.3 Lunettes

Des lunettes, avec un écran intégré ainsi que des accéléromètres, affichent la vue de la caméra du robot et permettent d'effectuer des mouvements de la tête.

Par manque de temps et de disponibilité, le programme n'a pas pu être testé avec des lunettes.

3 Modalités employées

3.1 Geste

Une grande partie de l'implémentation concernant les moyens utilisés pour la reconnaissance de gestes de la Kinect ainsi que la transmission par UDP (c++) jusqu'à l'implémentation de la réception des trames (c#) provient du travail d'approfondissement (Activity Recognition with Kinects) qui a été réalisé par Julien Tscherrig en parallèle au travail de MMI (Monik).

Pour les mouvements de la tête, les accéléromètres se situant dans le casque ont été utilisés. Il faut que l'utilisateur place sa tête horizontalement, car lors du lancement de l'application les coordonnées du casque se positionnent à 0. Ensuite à chaque mouvement, le casque envoie le delta entre la nouvelle position et celle d'avant. Pour déterminer l'angle à laquelle le robot place sa tête, on additionne simplement les coordonnées reçues et on envoie le résultat.

3.1.1 API et librairies disponible

Nous avons décidé d'utiliser le framework OpenNI avec le module NITE. OpenNi est un framework multimodales regroupant les modalités suivantes : reconnaissance de geste, reconnaissance vidéo et audio. Dans notre cas, OpenNI est uniquement utilisé pour la reconnaissance de gestes.

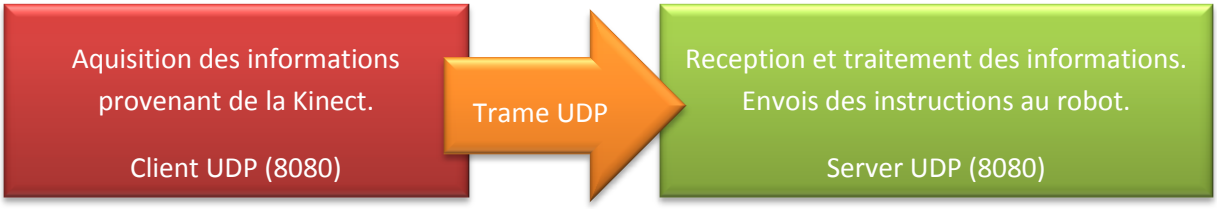
Actuellement, plusieurs librairies permettent de travailler avec une Kinect (OpenNI, OpenKinect…). Nous avons décidé de partir sur ce framework car il massivement utilisé par les développeurs utilisant une Kinect et dispose grâce au module NITE une reconnaissance et un tracking des personnes se trouvant devant la caméra.

- OpenNI Unstable Windows x86 (32-bit) v1.1.0.41 Dev Edition
- PrimeSensor Module Unstable Windows x86 (32-bit) v5.0.1.32
- PrimeSense NITE Unstable Windows x86 (32-bit) v1.3.1.5 Dev Edition

3.1.2 Architecture

Afin de facilité son implémentation et d'offrir un moyen simple de travailler avec notre programme final écrit en c#, nous avons décidé de partir sur le programme déjà réalisé par Julien Tscherrig écrit en c++ permettant de récupérer les informations liées à la Kinect.

La communication entre le programme c++ et c# se fait à l'aide d'une communication UDP.



3.1.3 Implémentation pour l'acquisition (client)

Afin de gagner du temps, le programme charger de récupérer les informations provenant de la Kinect (détection et tracking des personnes) est basé sur l'exemple NIUserTracker (c++) proposé en tant qu'exemple lors de l'installation d'OpenNI.

Ils nous a suffi de le modifier légèrement pour lui rajouter un simple client UDP/IP qui va se charger d'envoyer une trame au format XML contenant les informations vue par la Kinect lors de chaque nouvelle acquisition d'image 3D sous forme de matrice de profondeur.

```
<skeleton>
    <skeleton_part part_id="1" x="102" y="482" z="1233"
    <skeleton_part part_id="5" x="202" y="422" z="1673"
    <skeleton_part part_id="6" x="122" y="532" z="237"
    <skeleton_part part_id="7" x="652" y="132" z="4233"
    ...
</skeleton>
```

3.1.4 Implémentation pour l'acquisition (serveur)

La réalisation du programme permettant le contrôle du robot étant réalisée en c# et non en c++, doit permettre la réception de ces fameuse trame au format XML.

3.1.4.1 Enregistrement des trames réceptionnées

Afin de pouvoir procéder par la suite au traitement des trames, il est nécessaire de pouvoir les enregistrer quelque part dans le programme pour y avoir accès plus tard. Il s'agit du rôle du KBStore et de ces classes héritées se trouvant dans le package Monik.Model.

Le KBStore offre un moyen de stockage rapide et performant mais sans aucune traitement sur les données. Une couche KBLayerIntelligency est héritée du KBStore et offre des moyens de traitement au niveau des trames, dont la possibilité d'effectuer des tris au niveau des trames réceptionnées à l'aide du TrameIdentificator.

Le constructeur du KBStore nécessite une classe implémentant l'interface Update. Cette instance Updater sera informée à l'aide de la méthode Update qu'elle implémentera des toutes nouvelles arrivées de trame.

```
public KBStore(Updater instanceUpdater)
public KBLayerIntelligency(Updater instanceUpdater)
```

A l'intérieur du programme réceptionnant les trames (c#), elles sont traitées à l'aide de la classe TrameInfo présente dans le package Monik.Type.Trame. Cette classe permet le découpage et traitement des trames envoyées au format XML et TramePart qui contient les coordonnées sur chacun des joints du squelette présents.

3.1.4.2 Serveur UDP/IP

La réception de trames se fait à l'aide du serveur UDP se trouvant dans le package Monik.Logical.Server soit les classe ThreadReceiver et TrameReceiver. Pour rendre le serveur de réception actif, il suffit de construire un TrameReceiver de lui indiquer le numéro du port sur lequel

le serveur doit écouter (par défaut, il s'agit du 8080) et d'en indiquer le KBStore soit la classe permettant de stocker les trames réceptionnées.

Constructeur de la classe initialisant la réception des trames provenant du client UDP/IP appartenant au programme réalisant l'acquisition des informations provenant de la Kinect

```
public TrameReceiver(int portReceiver, KBStore dataStore)
```

Extrait du code permettant la réception à l'aide du serveur UDP disponible dans la classe ThreadReceiver

```
public void doWork(){
    while (true){
        byte[] data = new byte[BUFFER_SIZE];
        EndPoint tmpRemote = (EndPoint)(sender);
        int recv = newsock.ReceiveFrom(data, ref tmpRemote);
        string xmlString = Encoding.ASCII.GetString(data, 0, recv);
        TrameInfos trame = new TrameInfos(xmlString);
        dataStore.addNewTrame(trame);
    }
}
```

3.2 Voix

Permet de déplacer le robot en utilisant les commandes suivantes : avancer, reculer, gauche, droite et stop.

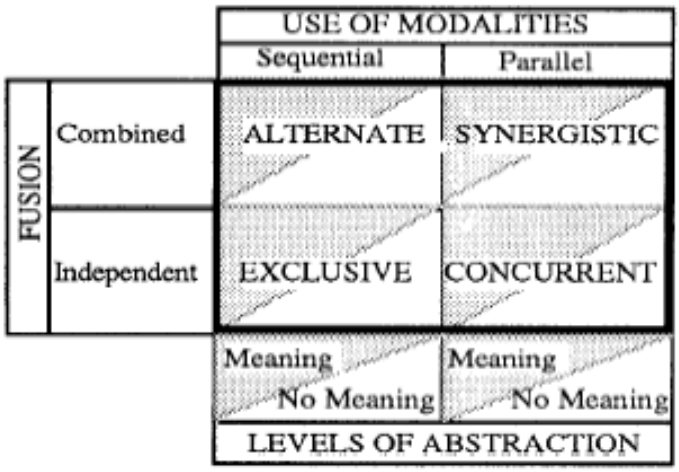
La librairie employée, disponible directement dans le Framework 3.5 en *.NET*, est *SpeechRecognition*. Elle est très facilement utilisable et obtient d'excellents résultats. Le principe de fonctionnement est très simple, on crée un dictionnaire (aussi appelé grammaire) contenant la liste des mots qu'on veut reconnaitre. Ensuite on crée l'objet qui permet de reconnaitre les mots et on ajoute un « event handler » qui sera appelé à chaque fois qu'un mot est reconnu. Voici un exemple :

```
public SpeechRecognition(){
    // Création du dictionnaire des mots
    GrammarBuilder grammar = new GrammarBuilder();
    Choices action = new Choices("avance", "recule", "stop");
    grammar.Append(action);

    // Initialisation de la reconnaissance vocale
    SpeechRecognitionEngine sre = new SpeechRecognitionEngine();
    sre.LoadGrammar(new Grammar(grammar));
    sre.SetInputToDefaultAudioDevice();
    sre.RecognizeAsync(RecognizeMode.Multiple);

    // Ajout d'un gestionnaire d'évènements
    sre.SpeechRecognized                    +=                    new
    EventHandler<SpeechRecognizedEventArgs>(_SpeechRecognitionRecognized);
}

void _SpeechRecognitionRecognized(object sender, SpeechRecognizedEventArgs e){
    //Affichage du text reconnu
    Console.Out.WriteLine(e.Result.Text);
```

4 Modèles CARE / CASE

4.1 CASE

Le modèle CASE (Concurrent, Alternate, Synergisitc, Exclusive) est un concept pour catégoriser les applications multimodales d'un point de vue machine. D'après ce modèle nous avons une application multimodale **concurrente**. Nos modalités, la voix et le geste, peuvent en effet être utilisées simultanément, mais leurs rayon d'action ne touche pas les même fonctionnalités, elles sont indépendantes.



4.2 CARE

Le modèle CARE (Complementarity, Assignment, Redundancy, Equivalence) a le même objectif que le modèle CASE, mais se positionne d'un point de vue humain pour catégoriser les applications multimodales. Nous avons développé une application de type **assignment**, car une seule modalité permet d'atteindre un objectif.

Il est possible de faire avancer le robot seulement en utilisant la voix, aucune autre modalité ne permet de le faire.

5 User evaluation

Cette phase d'évaluation vise à répondre à deux questions qui se posent à propos de l'utilisation des modalités de geste et voix pour contrôler un robot : le principe de mimétisme est-il pertinent, naturel pour l'utilisateur et l'utilisation de la voix pour contrôler les déplacements du robot est-elle un handicap ?

Premièrement, nous avons asserté en début de projet que l'utilisation du mimétisme pour contrôler les gestes des bras et de la tête du robot était la manière la plus naturelle pour l'utilisateur. Nous avons tenté de répondre à cette question par le biais d'une série d'essais par différentes personnes n'ayant pas participé au projet. Ces essais sont observés par l'équipe de développement afin d'évaluer le confort d'utilisation. S'ensuit une discussion avec chaque utilisateur afin d'avoir un feedback de leurs impressions. Cette phase de l'évaluation nous a permis de prouver que le mimétisme est effectivement un moyen très naturel de pilotage d'un dispositif humanoïde. La prise en main du contrôle des bras et de la tête est instantanée, elle est un peu plus laborieuse pour les commandes vocales qui sont moins naturelles. Néanmoins, le confort d'utilisation est entravé par les limitations techniques du robot et le délai engendré par la reconnaissance des commandes vocales. En effet, le mouvement des bras du robot est restreint par rapport à celui d'un être humain ; il ne peut par exemple pas joindre les mains ou croiser les bras. Le temps de réaction du robot dans ses déplacements lors d'un ordre vocal est également trop long et cela rend les manœuvres très difficiles.

Cette première phase d'évaluation nous a permis d'avoir des débuts de réponses à nos questions. Afin d'avoir des résultats plus fiables, il serait intéressant d'avoir une deuxième phase d'évaluation sous la forme du suivi d'un parcours fixe dans des conditions de laboratoire avec des mesures de temps pour aller du début à la fin, en effectuant des tâches données sur le long du parcours. Il serait également intéressant d'implémenter une solution alternative aux commandes vocales pour effectuer les déplacements du robot, par exemple un système de pseudo-mimétisme ou l'utilisateur devrait marcher « sur-place ». Ceci est malheureusement peu envisageable avec les moyens de détection que nous avons.

Cette seconde partie de l'évaluation ne sera malheureusement pas faite parce qu'elle demanderait des moyens et du temps que nous n'avons pas.

6 Conclusion

L'idée de ce projet était d'explorer les possibilités de guidage d'un robot *NAO* par le principe du mimétisme via le dispositif *Kinect* et des *accéléromètres*. Nous avons également utilisé les commandes vocales pour effectuer les actions qui n'étaient pas faisables par le mimétisme, comme les déplacements du robot.

En l'état actuel, la commande du mouvement des bras et de la tête, respectivement par la *Kinect* et les *accéléromètres*, fonctionne très bien. Les déplacements du robot (avancer, reculer, à gauche, à droite et stop), effectués grâce aux commandes vocales, fonctionnent également. Le retour visuel, la caméra du robot, est elle-aussi affichée correctement dans les lunettes de réalité augmentée.

L'évaluation par les utilisateurs nous a démontré que notre but était atteint, avec un petit bémol au niveau de la précision atteinte lors de l'imitation des mouvements. Ceci est principalement dû à la cumulation de la marge d'erreur des différents dispositifs : la détection du squelette de l'utilisateur par la *Kinect*, la précision des accéléromètres, ainsi que le temps de réaction du robot et le changement d'échelle due à sa taille. Il ne serait pas imaginable d'utiliser ce robot dans une situation réelle car ses mouvements sont relativement limités, mais il faut garder à l'esprit qu'il s'agit d'un appareil destiné à la recherche et pas d'un outil professionnel.

Si ce travail vient à être poursuivi, il serait intéressant de trouver un moyen de commander les trois doigts du robot. Ceci n'est pas faisable par la détection du squelette via la *Kinect*, cette dernière ne détectant pas la position des doigts de l'utilisateur. Il serait également utile de pouvoir contrôler les déplacements du robot par pseudo-mimétisme, c'est-à-dire en imitant le mouvement de marche sur place. Il est en effet difficilement envisageable de se déplacer réellement dans la pièce, à moins d'avoir un système de détection composé de plusieurs *Kinect*.