DELBERGHE Calvin
SUEUR Florent

Interaction via Kinect sur un rayon commercial

Rapport de projet de quatrième année
Département IMA

Sommaire

Introduction

Le projet "*Interaction via Kinect sur un rayon commercial* " est un projet qui fait suite au projet abordé l'année passée sous le nom de "*Interaction gestuelle en environnement de vente*". Nous n'avons pas hésité à le reprendre pour notre projet de quatrième année à l'école d'ingénieur Polytech'Lille au sein de la filière IMA, option Systèmes communicants (Informatique et Electronique) pour y apporter des améliorations.

La Kinect a attiré depuis sa sortie l'attention d'innombrables développeurs, et tout particulièrement récemment, depuis que le SDK Kinect pour Windows est sorti.

La Kinect est un nouvel intermédiaire entre l'homme et la machine qui permet de revisiter l'interaction entre ceux-ci. Elle donne aux développeurs la perspective d'inventer de nouvelles applications que la science-fiction avait déjà abordé.

Dans un premier temps, nous présenterons les divers outils et périphériques que nous avons utilisés tout au long de ce projet. Puis nous expliquerons les évènements les plus importants intervenus lors de celui-ci. Ensuite, nous reviendrons plus en détail sur l'interface que nous avons mis au point ainsi que sur le fonctionnement du programme principal qui gère la Kinect. Et enfin, nous terminerons par les améliorations envisageables pour d'éventuels futurs projets.

Remerciements

Nous voulions remercier tout particulièrement tous ceux qui nous ont soutenus, aidés et permis de réaliser jusqu'au bout notre projet de 4ème année, en particulier :

- Monsieur Laurent Grisoni, Maître de conférence à Polytech'Lille et encadrant de notre projet pour son aide et ses conseils tout au long du projet ainsi que pour l'autonomie formatrice qu'il nous a laissé.
- Monsieur Xavier Redon, Maître de conférence en informatique à Polytech'Lille pour l'aide à l'installation d'OpenNi ainsi que pour avoir facilité la portabilité de notre projet vers la Télévision.
- La société Oxylane pour l'intérêt porté à ce projet.
- L'ensemble de nos camarades présents en salle de projet pour avoir bien voulu tester notre application et nous avoir montré des erreurs que nous n'aurions pas vu sinon.

I. Description des outils utilisés

a) Kinect

La Kinect est un périphérique sorti pour le grand public début Novembre 2010. A la base nommé Project Natal, ce produit fut développé par Primesense dans un premier temps puis commercialisé par Microsoft comme périphérique de contrôle de jeux pour la console Xbox360.

La Kinect est un appareil composé de multiples capteurs: une caméra classique, une caméra et un émetteur infrarouge, des microphones pour la reconnaissance vocale et un moteur dans la base permettant de suivre les mouvements de l'utilisateur. Les plus intéressants sont bien entendu la caméra et l'émetteur infrarouges qui permettent de distinguer la profondeur de ce qui est dans le champ de vision de la caméra classique.

Cependant la Kinect n'a toujours pas atteint son plein potentiel, les jeux n'utilisent que 20 à 30 % des capacités réelles de cet appareil. En effet seules les fonctionnalités basique de cet appareil sont exploitées dans les jeux: problème de la résolution d'image petite afin d'avoir un nombre d'images par seconde correct, impossibilité de distinguer les mouvements des doigts d'une main, problème du nombre d'utilisateurs possible de distinguer à l'écran.

Il est évident qu'à la sortie de cet appareil les développeurs du monde entier ont réalisé le potentiel immense de cet appareil et ont commencé à l'utiliser pour des usages différents que ceux proposés initialement par Microsoft. Afin de couper court aux multiples hacks de l'appareil et aux développements de drivers non officiels, PrimeSense a mis à disposition du monde entier des drivers Open Source en Décembre 2010, ceux de l'API OpenNI. Ceux-ci étaient complémentés par l'API NITE qui augmentait encore les possibilités de l'appareil.

b) OpenNi

OpenNI vient de Open Natural Interaction, en effet cette API permet de concevoir des applications qui mettent en jeu les interactions naturelles, c'est à dire tous les gestes que nous pouvons faire lorsque nous communiquons avec une autre personne. OpenNI permet jusqu'à un certain point d'interagir avec une application de la même manière grâce à des capteurs.

OpenNI permet en effet la communication entre des capteurs réels (tels qu'une caméra RGB, des caméras / émetteurs infrarouge afin de distinguer la profondeur, des micros..) et des "percepteurs" qui sont en fait les modules de OpenNI qui vont traiter les données acquises par les capteurs et en tirer des informations utiles à l'application développée.

Ces percepteurs permettent de traduire par exemple la détection de mouvements d'un utilisateur, de suivre les mouvements d'une main, de reconnaître la voix d'un utilisateur ou même des ordres provenant de celui-ci. Ils peuvent détecter où se trouve le sol ou encore récupérer le fond derrière les utilisateurs. Ils permettent aussi de détecter les utilisateurs et de leur assigner un numéro et une couleur, et ensuite d'accéder à des informations comme le centre de gravité de l'utilisateur ou les coordonnées des pixels qui représentent l'utilisateur sur l'image.

OpenNi permet de coder ces applications dans 4 principaux langages: en C++, C#, Java voire en C. Nous avons choisi le C++ pour des raisons pratiques, tout d'abord parce que c'est un langage plus orienté objet ce qui simplifie grandement le code, et ensuite parce que nous ne connaissions ni le C# ni le Java assez bien pour pouvoir coder correctement dans ces langages. De plus tous les exemples fournis étaient codés en C++ ce qui n'était pas le cas pour les autres langages.

c) NITE

NITE est une API qui, combinée à OpenNI, permet d'aller encore plus loin dans les interactions homme/application et offre de nouvelles possibilités non disponibles dans OpenNI seule. Il s'agit en effet d'un jeu de modules pour OpenNI qui permettent d'apporter des contrôles plus élaborés aux percepteurs de OpenNI.

NITE permet par exemple d'assigner un squelette basique (une quinzaine de points comprenant la tête, le torse, les membres inférieurs et supérieurs) à un utilisateur détecté par les capteurs de la Kinect. Cependant NITE n'assigne pas un squelette au hasard, il y a deux solutions pour assigner la bonne taille de squelette à un utilisateur donné:

- Soit l'utilisateur doit se mettre dans une position de calibration, appelée "psi-pose", qui permet au programme de bien différencier toutes les articulations où il va placer les points du squelette (exemple de psi-pose et du squelette en Annexe 1).
- Soit si le développeur l'a programmé, NITE peut aller chercher un squelette préenregistré dans une base de données en fonction de la taille et de la largeur apparente de l'utilisateur (informations récupérées grâce aux fonctions basiques de OpenNI).

Ce qui nous a particulièrement intéressés durant notre projet est la reconnaissance de gestes simples de la main intégrée à NITE:

- Push: Lorsque l'utilisateur fait un mouvement de poussée vers l'avant avec la main .
- Wave: Consiste à faire un va et vient avec la main de gauche à droite deux fois.
- Swipe: La simple reconnaissance du mouvement de la main dans les 4 directions du plan de l'image, haut, bas droite et gauche.
- Circle: Comme son nom l'indique, reconnaît lorsque l'utilisateur dessine un cercle avec la main.
- Steady: Le dernier mais le plus important de tous, il permet de reconnaître lorsque l'utilisateur a fini de faire un mouvement et est revenu en position initiale. Bien utilisé il autorise l'envoi d'un signal de la forme que l'on souhaite à l'utilisateur afin de l'informer qu'il peut procéder au mouvement suivant.

d) QT Creator

QT Creator est un IDE (Integrated Development Environment), un programme regroupant un ensemble d'outils pour le développement de logiciels. En effet QT Creator possède un éditeur de texte, un compilateur de C++ et QML, un débogueur ainsi que des outils de fabrication automatique.

Ce qui nous a particulièrement intéressés dans cet outil est le fait qu'il permet de créer des interfaces graphiques facilement, compilables et lançables comme des programmes C++. Pour coder ces interfaces il nous a fallu utiliser le langage QML. Le langage QML est du type déclaratif, c'est à dire qu'il s'agit de déclarations de propriétés permettant de mettre en forme les éléments de l'interface, comme par exemple la position dans la fenêtre, l'image qu'il faut faire apparaître, ou encore le dimensionnement de l'élément.

II. Déroulement du projet

a) Installation d'OpenNi et NITE et tests des samples de NITE

L'installation d'OpenNi et de NITE s'est passée sans trop de difficultés, c'est pourquoi nous sommes passés rapidement à l'étude et au test des exemples fournis par OpenNi, et plus particulièrement ceux de NITE :

- **Sample-Boxes** : Cet exemple permet d'apprendre à utiliser l'objet SelectableSlider1D, qui permet de récupérer la valeur d'un slider dans le sens de l'axe X ou de l'axe Z. Le geste Push était aussi utilisé.
- **Sample-SingleControl** est un petit programme qui est intégré dans les exemples de NITE mais que nous avons récupéré déjà modifié par un développeur sur internet. Ce programme permet de contrôler le curseur de la souris à distance via la Kinect. C'est ce programme qui nous a beaucoup inspiré. Il utilise l'objet NITE SelectableSlider2D qui utilise le même principe que SelectableSlider1D.

b) Difficultés rencontrées

Il est très difficile de reprendre un projet qui a déjà été abordé. Le code n'est pas toujours compréhensible dans l'immédiat, cela rend donc la perspective de l'améliorer d'autant plus difficile, sans compter le temps nécessaire à la compréhension du maniement des objets fournis par NITE et OpenNi. C'est pourquoi nous avancions plutôt lentement au début du projet. Heureusement, les exemples de NITE nous ont permis de trouver de nouvelles idées et nous avons repensé entièrement comment utiliser la Kinect dans un environnement commercial.

c) Une nouvelle interface pour la Kinect

Pour faire une nouvelle interface, nous avons beaucoup réfléchi aux outils que nous avions à notre disposition. Il était inenvisageable pour nous de choisir la SDL comme le projet de l'an passé. En effet, il aurait fallu charger une image pour afficher la liste des filtres, ainsi qu'autant d'images qu'il y aurait eu de filtre dans cette liste. Cela aurait rendu notre programme trop statique et il aurait été impossible pour un vendeur d'effectuer le moindre changement sans passer par la retouche d'une image.

L'apprentissage de la conception d'une interface graphique nous paraissait colossal compte tenu du temps qui nous était imparti. Nous nous sommes alors penchés sur QtCreator, qui permet essentiellement de créer des interfaces graphiques facilement. De plus ces interfaces étant écrites en C++, nous n'aurions pas eu de soucis de compatibilité de langage avec OpenNi et NITE qui sont eux aussi utilisables surtout en C++.

Nous avons alors découvert au sein du logiciel QtCreator le langage QML. Nous nous sommes donc inspirés des nombreux exemples regroupés dans QtCreator afin de créer une interface qui conviendrait à notre idée. Un d'entre eux a tout particulièrement retenu notre attention : Focus. Ce projet montre comment créer une interface simple interagissant avec les touches du clavier.

Projet exemple de Qt Creator: Focus

III. Description de l'interface

Comme il l'a été dit précédemment, nous avons utilisé le projet Focus comme une base pour notre interface, en le modifiant selon nos besoins.

a) Fichier principal

Le fichier principal (main.qml) permet de regrouper tous les éléments dont nous avons besoin :

- Une liste de plusieurs catégories sportives permettant à l'utilisateur d'affiner sa recherche d'articles et d'afficher ceux correspondant au filtre qu'il a choisi.
- Autant de grilles d'images qu'il y a de filtres dans la liste précédente.
- Un menu contexte, permettant d'afficher une aide ou des logos.

Voici comment est organisé ce fichier sous forme de schéma :

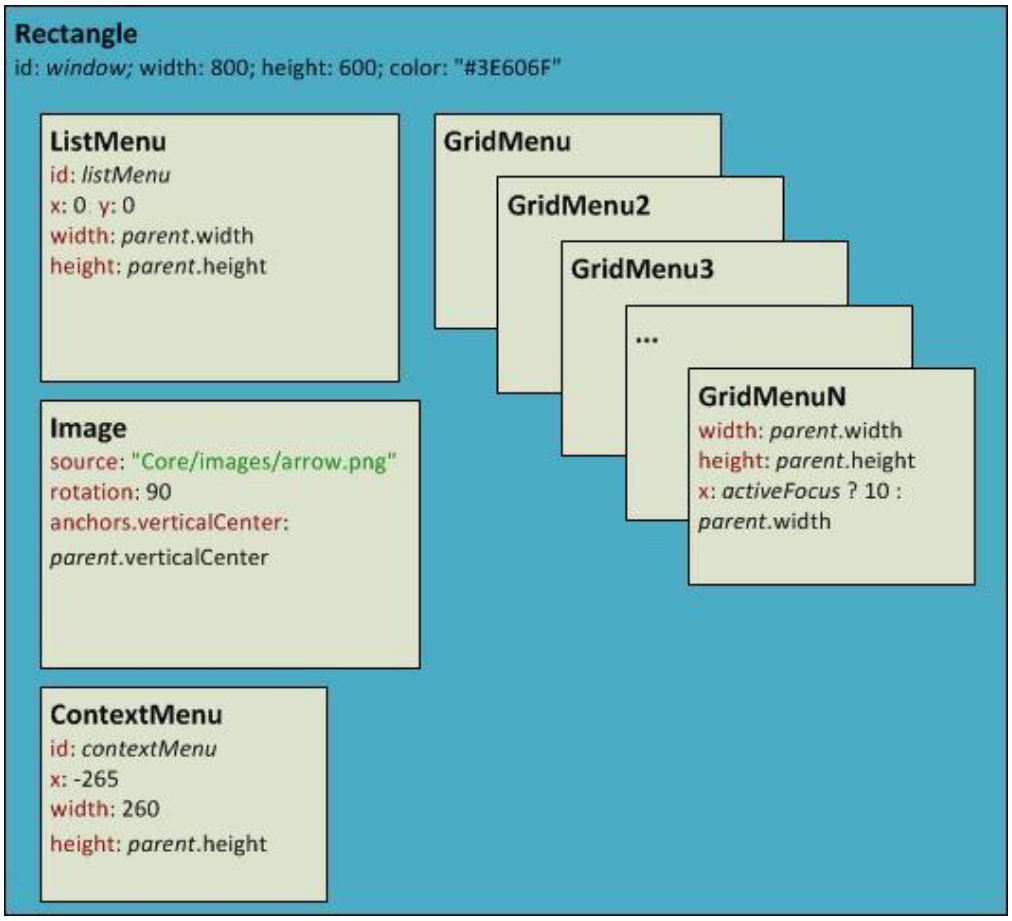
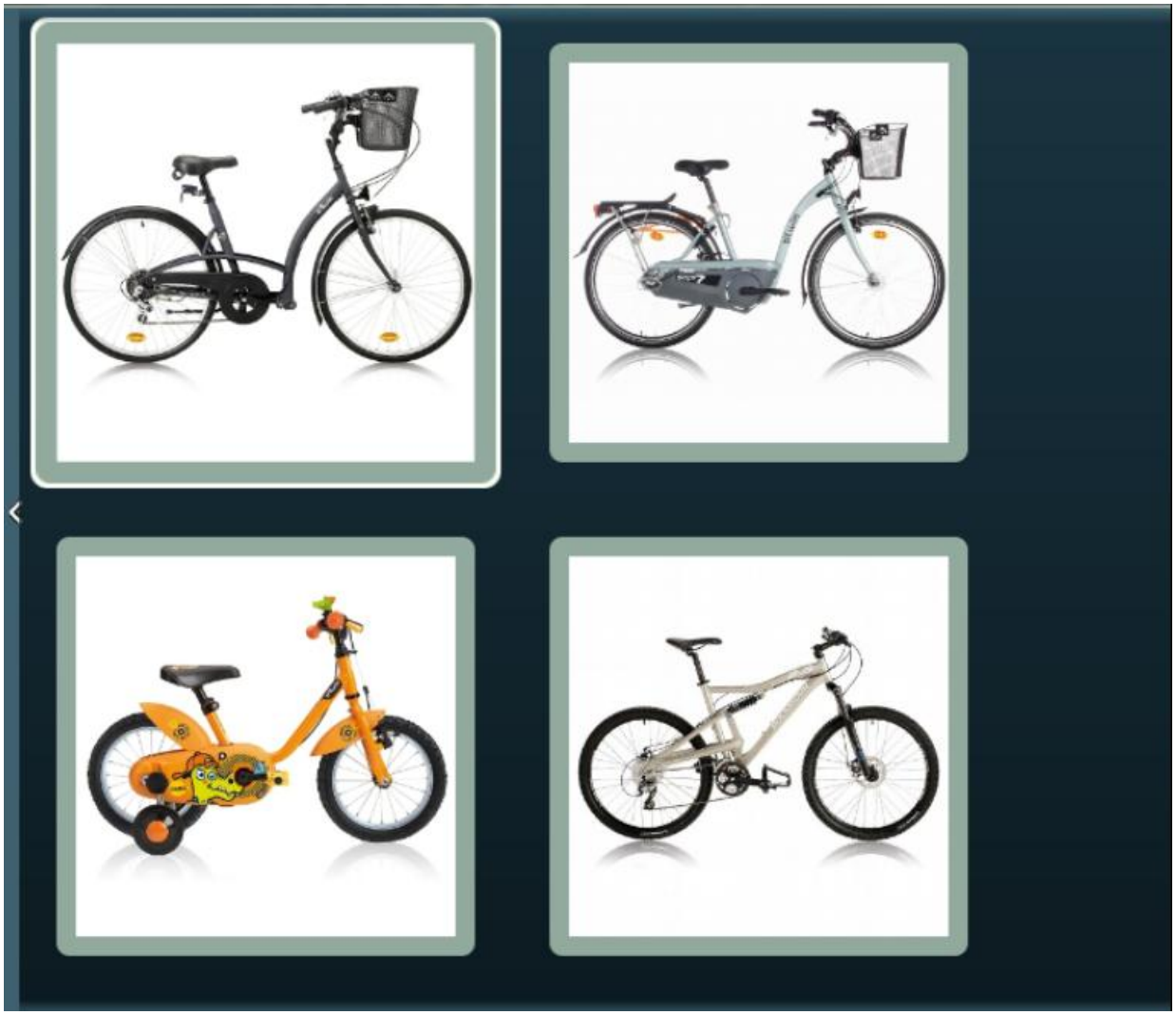


Schéma du fichier main.qml

b) GridMenu

Dans ce fichier, il y a un élément qui nous permet de placer les images sans avoir à se soucier de définir la position de chacune d'entre-elles individuellement, la *GridView*. Nous avons choisi une taille d'image suffisamment grande pour le confort de l'utilisateur. L'avantage est que nous n'avions pas à nous soucier de la taille de la fenêtre car les images s'étalent sur toute la surface de la fenêtre.

Il est plutôt facile d'ajouter de nouvelles images dans cette grille car il suffit d'ajouter un élément dans un fichier spécialement conçu pour contenir une liste d'articles. De plus, un des grands avantages du QML est qu'il n'est pas nécessaire de recompiler tout le projet après une modification.



Exemple d'une *GridView* d'images

c) ListMenu

Le fichier ListMenu.qml permet de référencer et d'afficher une liste de filtres pour que l'utilisateur puisse afficher les articles associés. Ici, on utilise un élément semblable à la *GridView* : la *listView.* Cet élément permet d'empiler ce que contient le fichier NomListe.qml qui contient une liste de chaines de caractères contenant les noms des filtres.

Exemple de liste

d) Interaction avec l'utilisateur

L'utilisateur peut naviguer dans cette interface graphique via les touches :

- Up, Down, Left, Right, ou plutôt les touches directionnelles.
- Tabulation.

Les touches directionnelles permettent bien sûr de se déplacer de haut en bas pour la liste, et dans toutes les directions pour les grilles d'images. Si on est à gauche de la grille d'image ou dans la page affichant la liste et que l'on appuie sur la touche de gauche, le menu contexte apparait. On quitte ce menu avec la touche de droite.

La difficulté avec le QML est que, comme avec le XAML ou le CSS, il n'est pas facile de faire des tests ou des switch/case. Nous avons donc utilisé certains évènements pour relier chacun des éléments de la liste avec sa grille d'images, notamment *OnActiveFocusChanged* qui est levé à chaque fois que l'on se déplace dans la liste. Ceci nous permet, selon si on est sur le n-ième élément de la liste, d'associer l'appui sur la touche tabulation avec l'ouverture de la bonne *GridView.* Dès que l'utilisateur veut voir les images du rayon "vélo", il appuie sur la touche Tabulation et accède à la bonne grille d'image. Pour retourner à la liste, il appuie à nouveau sur la même touche.

IV. Fonctionnement de l'application Kinect

a) Choix des gestes

Etant donné que nous avons une interface avec laquelle on interagit via les touches du clavier, nous nous sommes dit qu'il fallait des gestes simples, qui rappellent ces mêmes touches.

Il n'est pas évident de trouver des gestes que nous ne faisons pas inconsciemment lorsque l'on parle. Il faut donc éviter les gestes du type: se gratter la tête ou le bras pendant que l'on utilise le programme pour que la Kinect ne les prenne pas en compte comme une commande.

Heureusement, le projet NITE a très bien fait les choses. On a à disposition un panel de gestes qu'une personne ne peut pas faire inconsciemment.
Cela nous a par ailleurs évité le processus long et compliqué (faute de temps) qu'est celui de coder nous même les gestes via OpenNi.
Nous avons donc retenu ces gestes :

- *Push*, geste qui rappelle l'appui sur une touche de validation, ou Tab dans notre cas.
- *Swipe*, que nous avons décomposé en quatre gestes : *Swipe Left*, *Swipe Up*, *Swipe Right* ainsi que *Swipe Down*. Ces gestes rappellent assez bien les touches directionnelles du clavier, ce ne sont donc pas des gestes trop difficiles à retenir.
- *Circle*, mouvement de cercle de la main qui évoque un changement. Nous l'avons donc associé au changement d'interface, pour passer du projet de cette année, au projet de l'année dernière, et ceci dans les deux sens.
- *Steady*, qui permet comme nous l'avons expliqué précédemment, de reconnaitre le geste : pas de mouvement de la main. Nous l'avons utilisé pour reconnaitre quand la main de l'utilisateur ne bouge plus. Cela signifiera que l'utilisateur est prêt à faire un nouveau mouvement.

b) Simulation de l'appui sur une touche

Une fois que nous avions choisi nos gestes, il fallait trouver un moyen d'y associer l'appui sur une touche. Avec beaucoup de recherche, nous avons réussi à trouver un site montrant le code source d'une application montrant comment contrôler le curseur de la souris à distance via la main et la Kinect. (Voir Annexe 2)

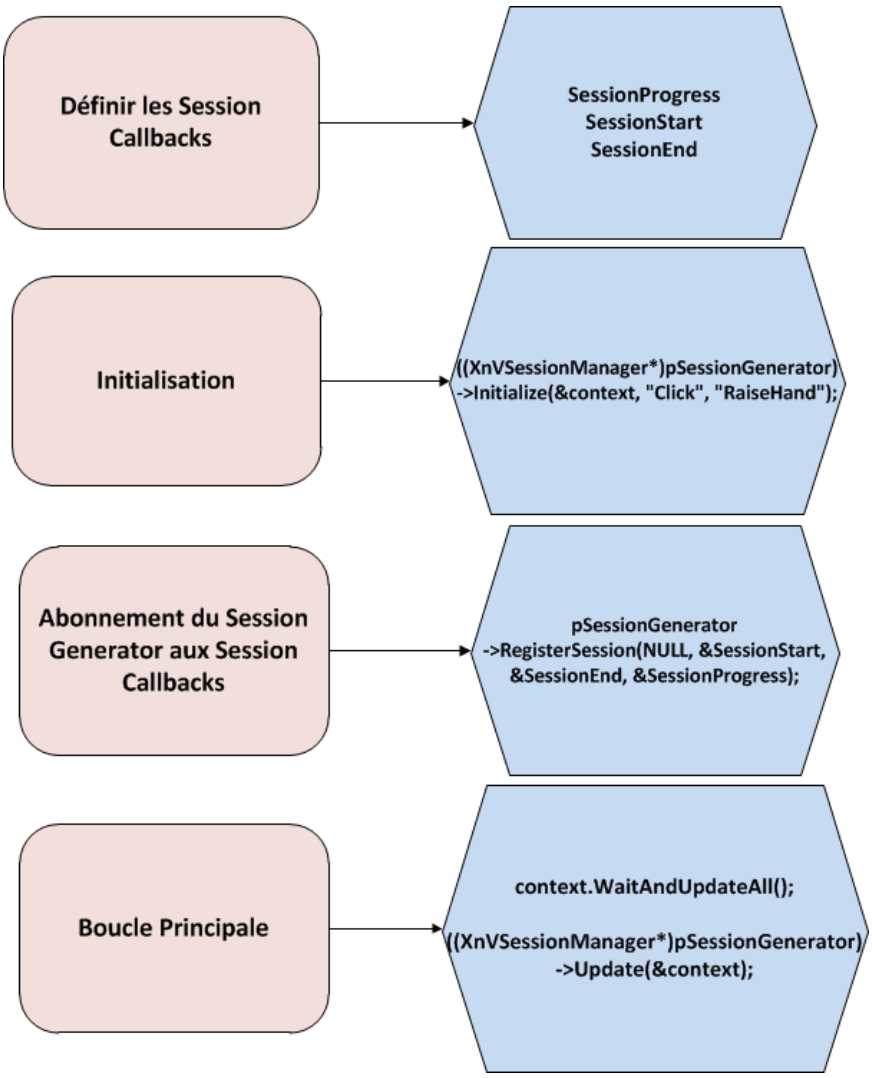
Nous nous en sommes inspirés pour simuler les appuis sur les touches dont nous avions besoin, car le principe est le même. (Voir Annexe 3)

c) Principe du programme

Pour commencer, il est utile d'expliquer ce que doit posséder au minimum un projet utilisant NITE.

Il faut tout d'abord définir les Session Callbacks, événements de début, fin de session et de session en cours. Nous les initialisons de telle sorte que lorsqu'un utilisateur réalise le geste *Push* (ou *Click*), la main est calibrée et suivie. Il faut ensuite faire en sorte que le programme écoute ces trois évènements, ainsi que tous les gestes dont nous avons besoin.

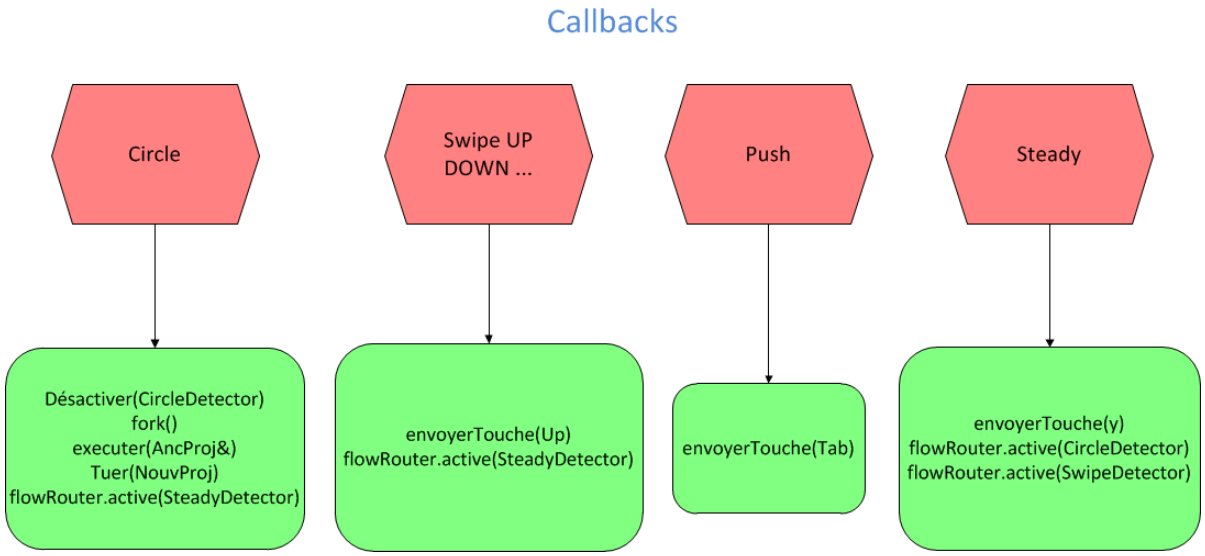
Dans le main, nous avons une boucle principale qui ne fait qu'attendre de nouvelles données en provenance de la Kinect.



Le minimum requis d'un programme NITE

Il faut maintenant définir les évènements qui nous intéressent et qui sont définis plus haut. Nous voulions associer la touche Tabulation avec le geste *Push*, les quatre gestes *Swipe* avec les touches directionnelles, le *Circle* qui permet de changer d'interface et enfin le *Steady* qui permet de détecter que la main de l'utilisateur ne fait aucun mouvement.

Voici l'algorithme qui est utilisé lorsque chaque évènement est levé:



A l'aide de ces évènements, nous pouvons donc facilement interagir avec l'interface que nous avons lancée au début du programme en tâche de fond pour ne pas empêcher la Kinect de détecter nos mouvements.

Nous avons cependant relevé un problème quand nous utilisions les gestes *Swipe*. En effet, nous ne savions jamais si la Kinect était prête ou non à reconnaitre un nouveau geste. C'est pourquoi, nous avons fait en sorte que quand la Kinect est prête à recevoir un nouveau geste, la touche "y" soit appuyée dans le programme principal. Côté QML, nous avons modifié notre programme pour que dès qu'il reçoive un appui sur la touche "y", il encadre en rouge l'élément courant de la liste des filtres ou de la grille d'images. Ainsi, l'utilisateur sait, par ce retour visuel, qu'il est autorisé à réaliser un nouveau geste.

Lors de nos premiers essais, nous n'avions pas utilisé de *FlowRouter*. Aussi, dès que nous réalisions un geste tel que *Circle* ou *Swipe*, l'événement était sans cesse levé jusqu'à ce que le geste *Steady* soit reconnu. Or si le code exécuté lorsque le mouvement *Circle* est détecté est répété plus d'une fois, alors il y autant de fermeture/ouverture des interfaces, ce qui est très gênant.

Nous avons donc approfondi nos recherches sur NITE et avons trouvé dans sa documentation (Voir Annexe 4) un moyen d'empêcher cela : le FlowRouter.

d) Le Broadcaster et le FlowRouter

Le *FlowRouter* permet de choisir quels évènements écouter. On peut ainsi créer facilement un cycle d'attente d'évènement. Nous avons choisi lors de l'initialisation de nos gestes d'attendre d'abord que l'utilisateur ait réalisé le geste *Push* pour reconnaître la position de sa main dans l'espace, puis d'attendre que l'utilisateur ne bouge plus.

Une fois que le programme a reconnu que l'utilisateur n'effectuait plus de mouvement, nous pouvons activer l'écoute des évènements *Swipe* et *Circle* et désactiver l'écoute du *Steady*. Lorsque l'un de ces deux gestes a ensuite été reconnu, nous désactivons leur reconnaissance et nous réactivons celle du Steady, et ceci en boucle infinie.

Enfin, il suffit de connecter le *PushDetector* et le FlowRouter à un broadcaster, pour que celui-ci redirige le flux qu'il reçoit à ses entrées au *SessionGenerator*, que nous avons initialisé plus haut. On peut donc faire un *Push* à n'importe quel moment.

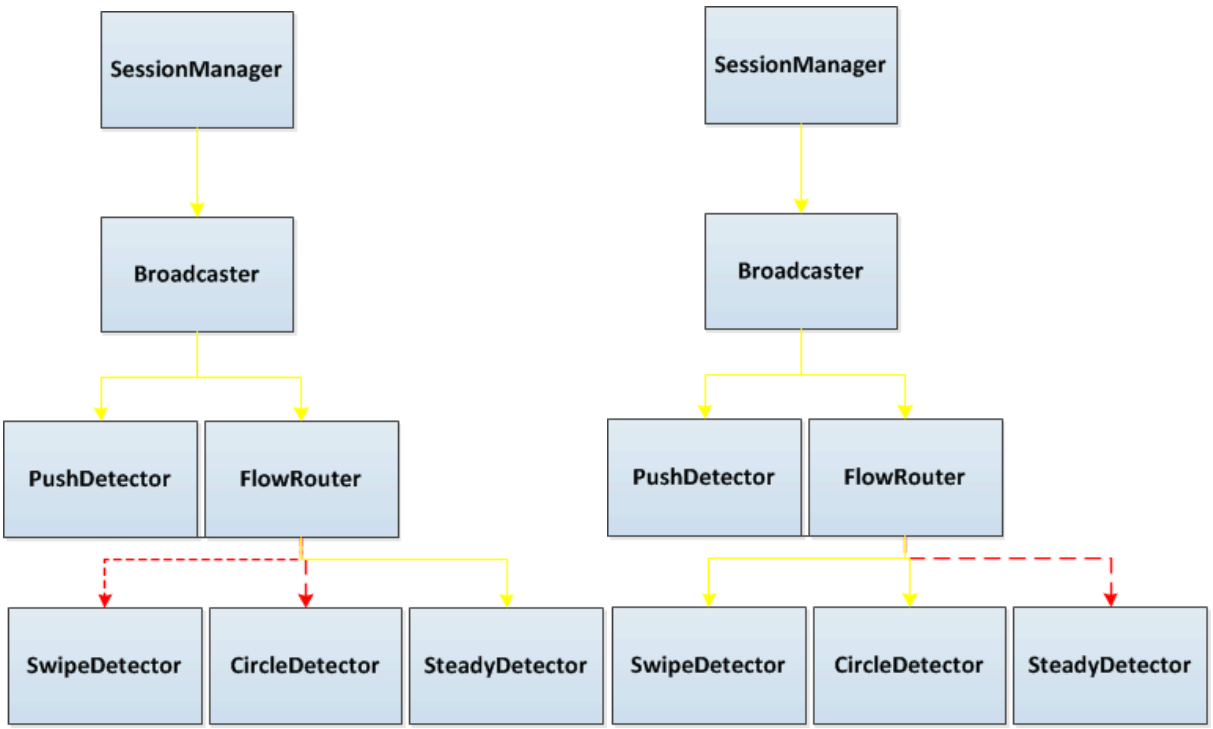


Illustration des connexions entre les objets NITE

V. Avantages et inconvénients du nouveau projet

Lors d'une reprise de projet, il semble normal de faire un petit comparatif des avantages et des inconvénients des deux projets. Premièrement, avec l'ancien projet, pour interagir avec l'application, il fallait absolument se placer sur une croix (un emplacement précis définit dans le code de l'application) pour pointer vers un objet de la "vitrine". Pour notre projet, nous n'avons pas besoin de respecter un emplacement précis. Il faut bien sûr se placer dans le champ de la caméra et à au moins un mètre. L'utilisateur peut donc se placer où il le désire pour réaliser le *Click* de début de session. Cette amélioration par rapport à l'an dernier semble moins contraignante au sein d'un magasin.

Deuxièmement, l'interface que nous avons développée est utilisable soit à distance, avec la main de l'utilisateur, soit directement avec le clavier, si le client ou le vendeur n'a pas envie de réaliser des gestes devant les autres personnes.

Troisièmement, étant donné que notre interface donne accès à un catalogue à l'utilisateur, il est tout à fait envisageable de placer la Kinect avec un écran en évidence dans un magasin. Notre programme n'a pas besoin d'un rayon physique derrière la Kinect pour viser des objets pour obtenir leurs caractéristiques. Il n'est pas envisageable économiquement de placer une Kinect avec un écran dans chaque rayon. Nous avons tout de même permis cette possibilité avec le geste *Circle* qui comme expliqué précédemment, lance l'interface d'interaction avec un rayon.

Un autre avantage est que l'utilisateur n'a pas à être nécessairement gaucher ou droitier. Une fois que la main est trackée, toute personne est capable d'interagir avec le programme quel que soit ses préférences.

Un dernier avantage est que notre interface est ajustable peu importe la résolution de l'écran. En effet celle-ci se met automatiquement en forme selon la place d'affichage disponible à l'écran grâce aux systèmes de *GridView* et *ListView* décrit précédemment.

Faute de temps, nous n'avons pas prévu que le programme arrête de tracker la main d'un utilisateur. C'est un inconvénient majeur du programme auquel il faudrait remédier.

Un autre inconvénient est que notre interface telle que nous l'avons définie ne permet pas d'afficher les caractéristiques d'un produit, ce qui était le cas dans le système précédent.

VI. Améliorations envisageables

Bien que le fonctionnement de notre programme final soit des plus satisfaisants, il serait possible de lui apporter des améliorations (plus ou moins importantes) pour le rendre plus efficace et/ou plus intéressant. Voici donc les améliorations auxquelles nous avons pensé tout au long du développement de notre système.

a) Modifications mineures

Attirer l'attention de l'utilisateur vers l'interface:

Avec le fait de placer ce système dans un rayon commercial réel vient la nécessité d'avoir des utilisateurs intéressés et attirés par le potentiel stand créé autour de la Kinect. Il serait donc judicieux d'ajouter dans le code une fonction qui lorsque la Kinect détecte un nouvel utilisateur à l'écran (fonctionnalité basique d'OpenNI), le programme lui envoie un signe, tel qu'une petite phrase ou une animation par exemple. Ceci permettrait d'attirer l'attention du client qui sait ce qu'il veut acheter, se dirige droit vers le rayon qui l'intéresse sans regarder autour de lui, et qui pourrait rater l'occasion de découvrir cette manière interactive de parcourir le catalogue du magasin.

Gérer plusieurs mains :

Le programme serait plus facile d'utilisation si il pouvait tracker les deux mains d'un utilisateur. NITE permet la gestion multi-main, il est donc facilement envisageable d'utiliser une main pour se diriger au sein de la liste et de la grille, et l'autre main pour valider.

Transitions plus fluides entre les différents gestes reconnus :

Actuellement, il faut attendre un certain temps pour que la Kinect soit prête à recevoir un nouveau geste après en avoir réalisé un. Nous avons essayé vers la fin du projet d'ajuster de notre mieux le temps pour la Kinect de reconnaitre le geste *Steady*. Cependant, soit le geste était trop long pour être reconnu, soit le geste était trop reconnaissable et il fallait réaliser les gestes *Swipe*, qui sont des gestes assez amples, rapidement, ce qui est gênant pour un utilisateur non habitué. Aussi, s'il serait intéressant d'améliorer la réactivité du geste *Steady*.

Afficher les caractéristiques d'un produit:

Telle qu'elle est faite, notre interface ne permet pas d'afficher les caractéristiques ni le nom (même si celui-ci est stocké dans la liste des produits) d'un produit en particulier dans une grille. En relation avec l'amélioration sur la base de données décrite plus bas, pour peu que l'on arrive à relier une base de données au projet, il suffirait de rajouter un geste qui permette d'accéder à un élément précis correspondant à une case d'une grille qui contiendrait et afficherait toutes les informations du produit en question.

b) Modifications majeures

Reconnaissance vocale:

Comme expliqué lors de la description de la Kinect, celle-ci possède pas moins de quatre microphones dont notre programme ne tire pas parti. Il serait intéressant d'intégrer cette fonctionnalité dans un futur projet. On pourrait par exemple utiliser une phrase du type "Kinect + commande" pour demander à un programme de réaliser une tâche particulière. Cependant, la reconnaissance vocale n'en est qu'à ses débuts et est en cours de développement. Autre inconvénient: il faudrait aussi passer à un développement en C#, et non plus C++, via le SDK Officiel de Kinect et utiliser le Framework .Net.

Mise en relation avec une base de données:

L'idée de ce projet était de proposer un système de présentation de produits pour un espace commercial. Ici nous avons mis en place un exemple de ce qui pourrait être fait, cependant celui-ci n'est pas exhaustif. En effet nous n'avons fait qu'ajouter quelques éléments du catalogue de Décathlon manuellement afin de montrer le concept de notre idée.

Le but serait de pouvoir mettre en place une relation avec une base de données de produits définis par des noms, des images et des caractéristiques. Il existe des solutions de mise en relation de bases de données et de code C++ déjà programmées (librairies qapplication.h, qsqldatabase.h et qsqlquery.h) fournies avec leur documentation disponibles en ligne. La difficulté serait d'adapter la base de donnée fournie par le client au programme afin qu'il autogénère les listes de produits dans notre interface.

Nouveau gestes:

Comme nous l'avons abordé dans les inconvénients du projet, nous n'avons pas eu le temps de prévoir la fonction permettant l'arrêt du suivi de la main de l'utilisateur. Il serait intéressant d'ajouter cette fonctionnalité en inventant un nouveau geste tel que le signe de la croix avec les bras (voir Annexe 5), comme le faisait le programme du projet de l'an passé, ou encore le geste *Wave*.

Agrandir une image avec les deux mains :

Cette amélioration est difficile dans le sens où il faut tout d'abord intégrer la fonctionnalité de la gestion multi-main. Ensuite on pourrait prendre deux SelectableSlider2D, un pour chaque main, pour modifier la propriété de la taille de l'image dans le QML.

Afficher du contenu selon la taille (déduction âge moyen):

Il serait intéressant d'afficher du contenu personnalisé à l'utilisateur tracké. On pourrait par exemple mesurer la taille d'une personne et selon si elle semble adulte ou enfant, afficher des vêtements, équipements ou jeux appropriés.

Conclusion

Ce projet nous fut confié dans le but de créer un système permettant d'interagir dans un milieu commercial de façon ludique et intuitive, mais on pourrait aussi imaginer des applications pour la vente d'immobilier, la publicité, ou encore dans la médecine, en permettant à un chirurgien de naviguer facilement entre les dossiers de ses patients à distance dans un environnement stérile.

Ce projet a été très formateur, en effet, il nous a permis de revoir la programmation événementielle abordée pendant le cours de conception logicielle objet. Il nous a aussi permis de nous rendre autonome autant pour l'apprentissage d'un SDK que pour la résolution de nos problèmes et le développement entier de notre programme.

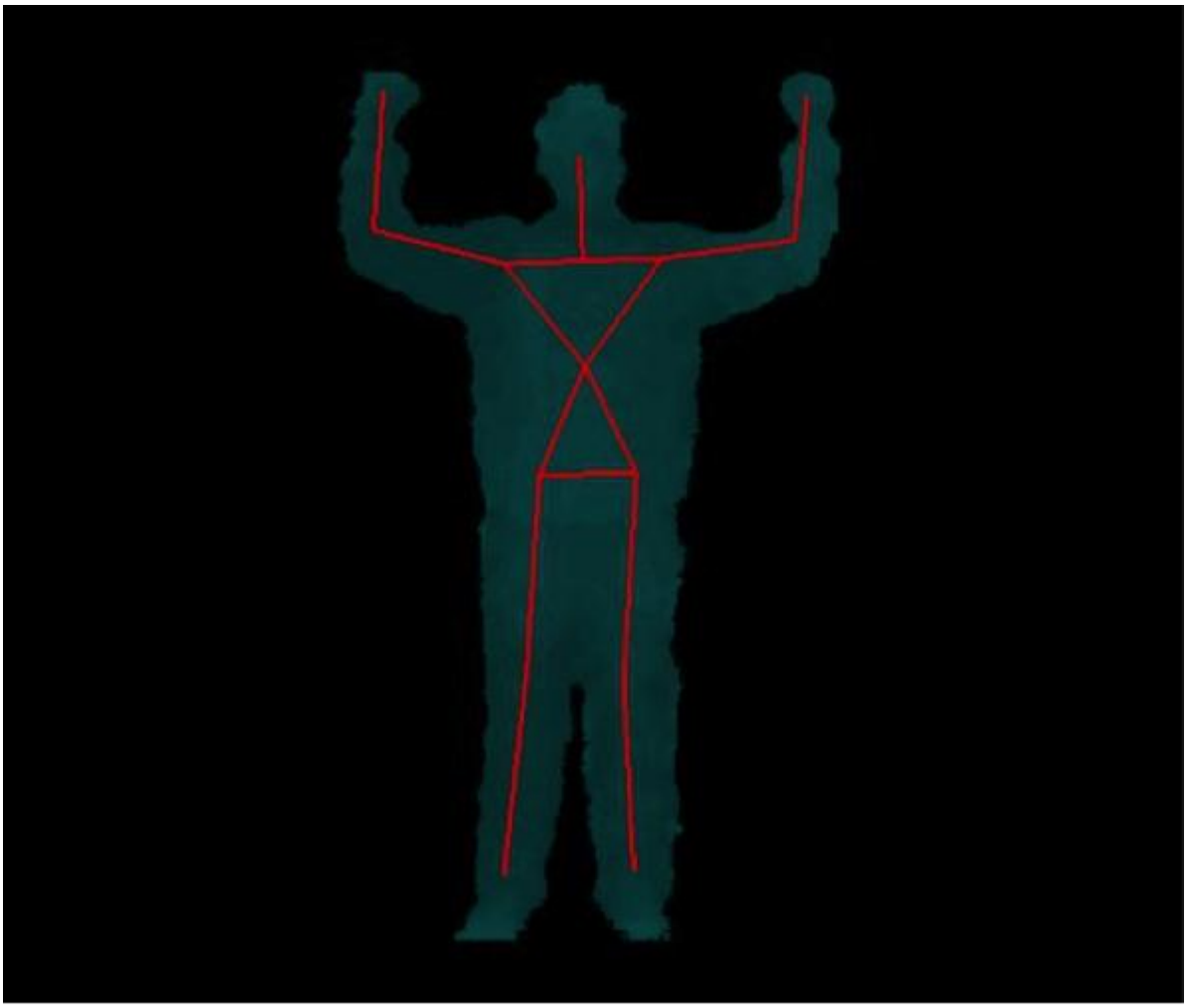
Les différents, et parfois nouveaux langages utilisés lors de notre projet nous ont permis de concrétiser et d'approfondir nos connaissances en Informatique, élément très important dans le cadre de nos projets professionnels

Nous avons dû faire face à une difficulté majeure, celle de la reprise d'un projet. Il est toujours plus facile de mener un projet dans son intégralité pour en maitriser toute les facettes pour ensuite l'améliorer. C'est pourquoi nous avons beaucoup détaillé les améliorations possibles avec des voies de résolution possibles.

Répartition du travail, respect des délais, auto-formation et travail en équipe sont des capacités qui nous seront utiles durant toute notre carrière.

ANNEXES

Annexe 1: Psi-Pose



Annexe 2: HOWTO: use the Kinect as a mouse in Linux

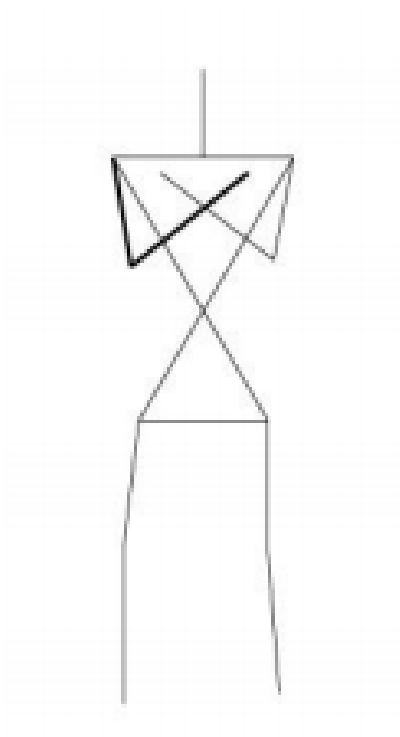
http://www.keyboardmods.com/2010/12/howto-use-kinect-as-mouse-in-linux.html

Annexe 3: Using uinput driver in Linux - 2.6.x to send user input

http://www.einfochips.com/download/dash_jan_tip.pdf

Annexe 4: Prime Sensor™ NITE 1.3 Controls Programmer's Guide
http://andrebaltazar.files.wordpress.com/2011/02/nite-controls-1-3-programmers-guide.pdf

Annexe 5 : Signe de croix avec les bras (tiré du rapport de l'an dernier)



Annexe 6: Rapport de projet de Florent Garrit et Morgan Mendes sur lequel nous nous sommes beaucoup appuyés au début du projet afin de mieux comprendre leur démarche et leur façon de coder
http://florentgarit.files.wordpress.com/2011/11/projet_kinect_mendes_garit_ima4sc_2012.pdf