Projet Virtual Robots

Julien GOETHAL et Adrien LIOTARD

2013-2014

Table des matières

Chapitre 1

Introduction

1.1 Contexte

Les robots sont de plus en plus présents : ils sont déjà utilisés pour les tâches domestiques, dans les entreprises, et pour aider les personnes âgées et/ou handicapées dans leur travail ou activités de la vie quotidienne. La collaboration humain-robot a jusqu'ici été étudiée dans le contexte d'applications industrielles avec un focus sur les aspects liés à l'acceptabilité et la sécurité. Au Japon les nouveaux robots industriels ont des caractéristiques anatomiques humaines dont l'objectif est de modifier les relations affectives avec les humains qui travaillent à leur côté. Les techniques de réalité virtuelle permettent d'immerger un humain dans un environnement 3D dans lequel il peut interagir et collaborer de manière naturelle avec des entités virtuelles ayant des formes diverses (humains, humanoïdes, robots, etc.). Par conséquent, cela permet de tester les réactions d'un humain face à des interactions avec différents robots sans avoir besoin de les acheter et de les installer. Ce projet a donc pour but de créer différentes activités entre un humain et un robot virtuel, pour ensuite analyser le comportement et le ressenti de l'utilisateur.

1.2 Cahier des charges :

Phase 1 : Développer un environnement virtuel et y intégrer différents types de robots
- modélisation de l'environnement
- modélisation des robots (apparence visuelle et squelette animable)
- intégration de l'environnement et des robots dans la plateforme logicielle

Livrable 1 : Description et rendu 3D de chaque robot et de l'environnement, et visualisation de l'environnement de travail dans Unity3D.

Phase 2 : Définir et simuler des tâches collaboratives entre l'utilisateur humain et le robot virtuel (manipulation d'objets par exemple)
- définition précise de la tâche (réactions correspondantes aux actions de l'utilisateur)
- développement des scripts logiciels adaptés
- tests utilisateur

Livrable 2 : Description de l'enchaînement des actions, scripts développés et description détaillée des tests utilisateur.

Phase 3 : Intégrer des interfaces de réalité virtuelle pour augmenter le réalisme et l'immersion (capteurs de mouvement LEAP Motion et Kinect, casque de visualisation Oculus Rift, dispositif de contrôle Razer Hydra).
Livrable 3 : Vérification du bon fonctionnement des interfaces de réalité virtuelle lors de l'exécution des tâches crées lors de la phase 2.

Phase 4 :Réaliser des tests utilisateurs :
- Évaluer l'influence de l'architecture et le comportement du robot sur le comportement et l'état émotionnel de l'humain
- Recueil de données subjectives (questionnaires) et objectives (mouvements de l'humain, signaux physiologiques, etc.)

Livrable 4 : Compilation et interprétation des données recueillies

1.3 Les outils utilisés :

1.3.1 Unity3D

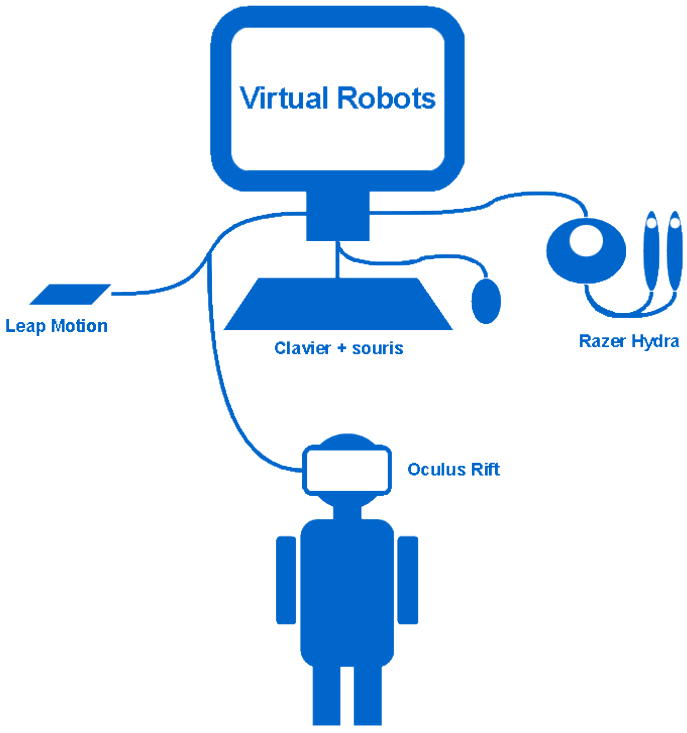
Unity3D est un moteur de jeux vidéo, c'est à dire un logiciel permettant de concevoir un environnement virtuel composé de modèles 3D (appelés "assets") contrôlés par des scripts. Par exemple pour un jeu de course automobile, un ensemble d'assets statiques servira à représenter le circuit (routes, barrières, panneaux, arbres, etc), tandis que les assets correspondants aux voitures seront commandés par des scripts. Ainsi la voiture pilotée par le joueur sera associée à un script réagissant aux entrées du clavier pour gérer la direction et la vitesse du véhicule, tandis que les adversaires seraient contrôlés par une intelligence artificielle.

Dans le cadre de notre projet, il n'y a pas d'adversaires mais des robots avec lesquels le "joueur" va pouvoir interagir. Mais contrairement à un jeu vidéo traditionnel, où l'on contrôle son avatar au moyen du clavier et de la souris (ou d'une manette) et où on voit le monde virtuel à travers un écran, ici l'objectif est de rendre l'expérience beaucoup plus immersive. En effet, l'utilisateur va effectuer ses actions au moyen de périphériques innovants (décrits plus bas), qui réagissent aux mouvements des mains ou de la tête et qui permettent une interaction plus naturelle avec les objets virtuels.

1.3.2 Blender

Blender est un programme de modélisation 3D. Il permet de créer des modèles en 3 dimensions à partir d'objets simples (cube, cylindre, sphère, etc) qui peuvent ensuite être modifiés au moyens de fonctions volumiques telles que l'extrusion, la rotation, la mise à l'échelle, ou directement en déplaçant les points du maillage correspondant au modèle. Grâce à Blender, il est donc possible de créer toutes sortes d'objets en 3D, qui pourront ensuite être importés dans Unity en tant qu'assets.

1.3.3 Équipements d'interface homme-machine :



Les périphériques utilisés

Leap Motion Le Leap Motion est un dispositif permettant de détecter les mouvements des mains. De la taille d'une grosse clé USB, il détecte très précisément la position et l'orientation des mains ainsi que celle des doigts,

au moyen de trois LED et deux caméras infrarouges (bien visibles sur le kit de développement ci-dessous). Les informations en provenance des caméras sont ensuite traitées selon un algorithme complexe (tenu secret par ses concepteurs), ce qui permet à l'ordinateur de "'voir"' les gestes effectués par l'utilisateur et ainsi de piloter divers programmes de façon plus naturelle et intuitive qu'avec une souris ou une manette.

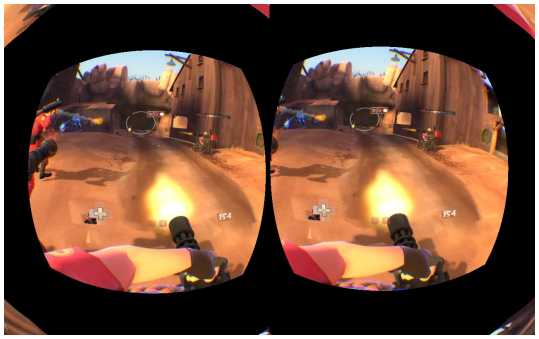
Leap Motion

Structure interne

Razer Hydra Le Razer Hydra est un contrôleur de jeu composé de deux manettes à détections de mouvements reliées à une base munie d'une bobine électromagnétique. Le maniement des manettes est très similaire à celui du Nunchuk équipant la console Nintendo Wii, et permet donc de nombreuses interactions avec un jeu vidéo : maniement d'une raquette ou d'une épée, reproduction des mouvements des mains, etc.

Razer Hydra

Oculus Rift L'Oculus Rift est un périphérique de réalité virtuelle se présentant sous la forme d'un masque recouvrant les yeux de l'utilisateur. Équipé d'un écran LCD ainsi que de deux lentilles, Il permet à l'utilisateur de voir en 3D un monde virtuel avec un grand champ de vision, ce qui permet d'avoir la sensation d'être présent dans le monde du jeu. D'autres part, divers capteurs détectant l'orientation de la tête renforcent encore ce sentiment d'immersion en adaptant dynamiquement l'image par rapport au regard du joueur.



Oculus Rift

Chapitre 2

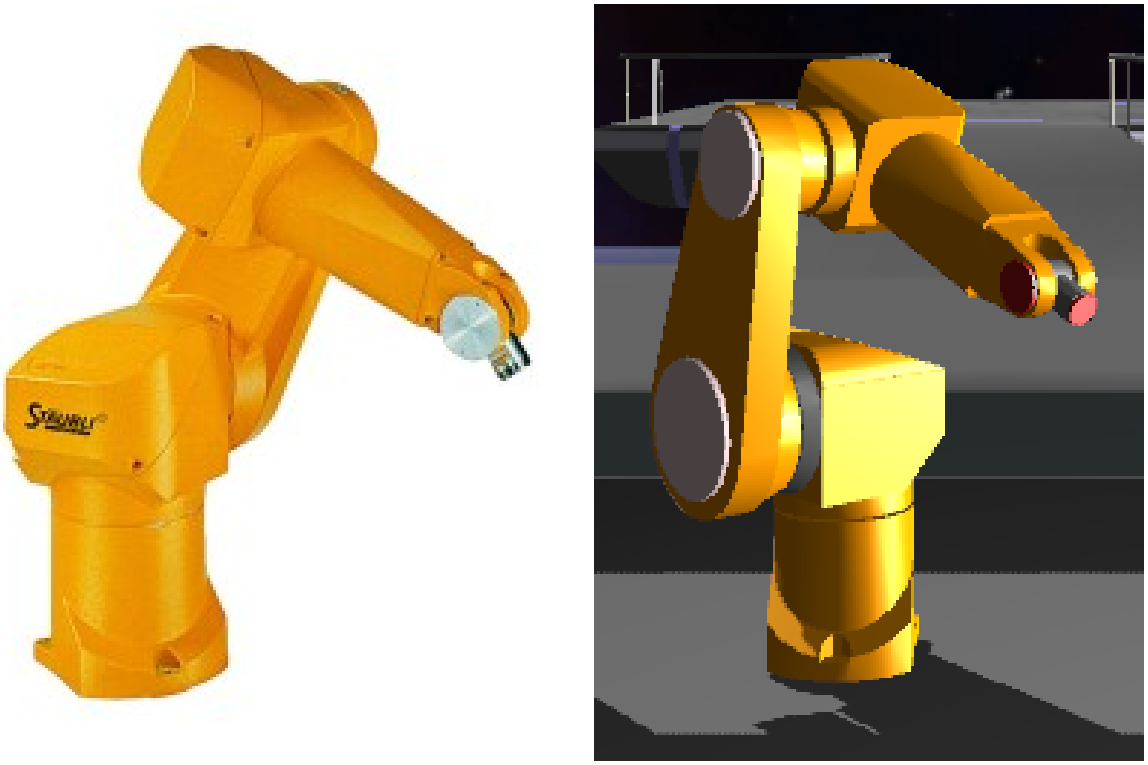
L'environnement virtuel

2.1 Le niveau

Dès le début, nous avons choisi de placer l'utilisateur dans un environnement futuriste, qui justifierait la présence des robots. Diverses pistes ont été explorées, en s'inspirant de certains films et jeux vidéos à l'architecture particulière (Oblivion, Portal,...). Finalement notre choix s'est porté sur le niveau "'Robots Lab"', présent sur l'Asset store d'Unity, ce qui a facilité son intégration dans le projet.

2.2 Un premier robot : le bras mécanique Staubli

Ce bras robotique industriel est pourvu de plusieurs servomoteurs qui lui confèrent 6 degrés de liberté et donc une grande mobilité. Notre choix s'est porté sur ce modèle car l'ISTIA en possède un exemplaire dans la salle de travaux pratiques d'automatique, et il serait donc envisageable de créer des interactions entre le modèle virtuel et réel.



Le bras robot Staubli

Le script ci-dessous a été l'un des premiers a être écrit lors du projet, il permet de contrôler les différentes articulations du bras avec les touches du clavier et les mouvements de la souris, ou encore en changeant la valeurs

des potentiomètres présents à l'écran. Ce script illustre bien la manière de procéder pour contrôler certaines parties d'un asset au moyen de code. En effet, les lignes 14 à 17 du script déclare des variables de type Transform. Ce type correspond à un objet 3D, qui est lié au script via l'Inspector d'Unity. Il est donc ensuite possible d'appeler la méthode Rotate sur chacune des parties du robot en fonction de certaines actions. Par exemple à la ligne 33 on déclare que la base du robot tourne selon l'axe Z en fonction des touches directionnelles gauche et droite du clavier. A la ligne 41, on indique le premier segment du bras tourne de 1 degré par frame si l'on appuie sur la touche "'a"'. Les commandes présentes dans la fonction OnGUI() permettent l'affichage des potentiomètres et la modification de la valeur correspondante à chaque articulation. Elles sont mises en commentaire dans le script car il n'y avait pas d'intérêt à se servir de potentiomètres commandés à la souris dans le projet final.

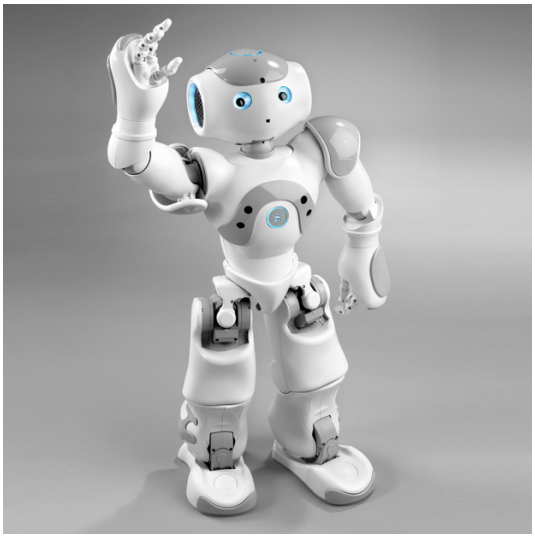
```
1  #pragma strict
2
3  // Draws a horizontal slider control that goes from -10 to 10.
4  var robotBaseSliderValue : float = 0.0;
5
6  // Draws a horizontal slider control that goes from -10 to 10.
7  var robotUpperArmSliderValue : float = 0.0;
8
9  var robotLowerArmSliderValue : float = 0.0;
10
11 var robotHeadSliderValue : float = 0.0;
12
13 //These slots are where you will plug in the appropriate robot parts into the inspector.
14 var RobotBase : Transform;//servo 1
15 var RobotUpperArm : Transform;//head
16 var RobotLowerArm : Transform;//virtual arm
17 var RobotHead : Transform;//virtual servo 2
18
19 //These allow us to have numbers to adjust in the inspector for the speed of each part's rotation.
20 var baseTurnRate : float = 1000;
21 var upperArmTurnRate : float = 5;
22 var lowerArmTurnRate : float = 1;
23 var headTurnRate : float = 1;
24
25 function Update(){
26
27 //Mathf.Clamp(RobotHead.rotation.z,-90,90);
28
29 //rotating our base of the robot here around the Y axis and multiplying
30 //the rotation by the slider's value and the turn rate for the base.
31
32 //RobotBase.Rotate (0, 0,robotBaseSliderValue * baseTurnRate);
33 RobotBase.Rotate (0, 0,Input.GetAxis ("Horizontal") * baseTurnRate);
34
35 //rotating our upper arm of the robot here around the X axis and multiplying
36 //the rotation by the slider's value and the turn rate for the upper arm.
37 RobotUpperArm.Rotate (0,Input.GetAxis ("Vertical") * upperArmTurnRate, 0);
38
39 //RobotLowerArm.Rotate (0,0,robotLowerArmSliderValue * lowerArmTurnRate);
40
41 if(Input.GetKey("a"))
42 {
43   RobotLowerArm.Rotate (1,0,0);
44 }
45
46 if(Input.GetKey("z"))
47 {
48   RobotLowerArm.Rotate (-1,0,0);
49 }
50
51 if((Input.GetKey("q"))&&(RobotHead.rotation.z<90))
52 {
53   RobotHead.Rotate (1,0,0);
54 }
55
56 if((Input.GetKey("s"))&&(RobotHead.rotation.z>-90))
57 {
58   RobotHead.Rotate (-1,0,0);
59 }
60
```

```
61 if(Input.GetKey("w"))
62 {
63   RobotBase.Rotate (0,1,0);
64 }
65
66 if(Input.GetKey("x"))
67 {
68   RobotBase.Rotate (0,-1,0);
69 }
70
71 RobotHead.Rotate (0,0,robotHeadSliderValue*headTurnRate);
72
73    if (Input.GetMouseButtonUp (0)) {
74
75         //resets the sliders back to 0 when you lift up on the mouse click down.
76         robotBaseSliderValue = 0;
77         robotUpperArmSliderValue = 0;
78         robotHeadSliderValue=0;
79         robotLowerArmSliderValue=0;
80     }
81
82 }
83
84 function OnGUI () {
85 /*
86 //creates the slider and sets it 25 pixels in x, 25 in y, 100 wide and 30 tall.
87 robotBaseSliderValue = GUI.HorizontalSlider (Rect (150, 25, 100, 30), robotBaseSliderValue, -10.0,
       10.0);
88
89 //creates the slider and sets it 25 pixels in x, 80 in y, 100 wide and 30 tall.
90 robotUpperArmSliderValue = GUI.HorizontalSlider (Rect (150, 80, 100, 30), robotUpperArmSliderValue
     , -1, 1);
91
92 robotHeadSliderValue = GUI.HorizontalSlider (Rect (150, 135, 100, 30), robotHeadSliderValue, -1,
     1);
93
94 robotLowerArmSliderValue = GUI.HorizontalSlider (Rect (150, 190, 100, 30),
     robotLowerArmSliderValue, -1, 1);
95
96 //GUI.Label (Rect (10, 10, 100, 20), RobotHead.transform.rotation.z);
97 */
98 }
```

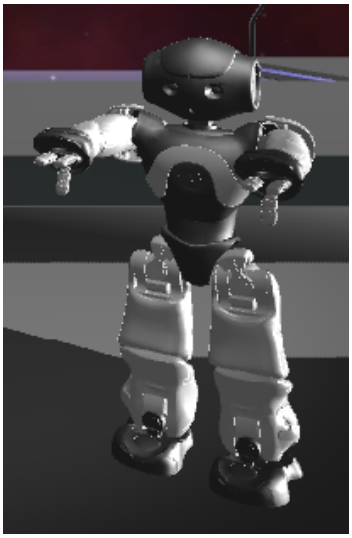
Listing 2.1 – robot control.js

2.3 Nao, le robot humanoïde

Mesurant 58cm pour un poids de 4,8Kg, Nao est un robot humanoïde très apprécié pour ses impressionnantes capacités de mouvement (marche, saisie d'objets, etc). il est notamment utilisé pour des compétitions de football robotique, ainsi que pour des chorégraphies complexes. Son apparence humanoïde lui permet d'attirer la sympathie des gens qui le voient en démonstration, à tel point qu'il est utilisé par certains chercheurs pour communiquer avec des enfants autistes. Pour toutes ces raisons, Nao semblait vraiment le robot idéal pour tester les relations humains-robots au cour de notre projet.



Nao en vrai



Nao dans Virtual Robots

```
#pragma strict

public var walkAnimation : AnimationClip;
private var _animation : Animation;
var Nao : Transform;
public var robotSound : AudioClip;

function Start () {

_animation = GetComponent(Animation);
for (var state : AnimationState in _animation) {
  state.speed = 4;
}

}

function Update () {

Nao.Translate(-Input.GetAxis ("Vertical")/10.0,0,0);
Nao.Rotate(0,0,Input.GetAxis ("Horizontal")*2);

 if (Input.GetAxis("Vertical")!=0 )
{
          _animation.Play();

        if (!audio.isPlaying){
        audio.clip = robotSound;
        audio.Play();
        }
}
     else
{
          _animation.Stop();
}
}
```

Listing 2.2 – nao walk.js

Chapitre 3

Jeux et tâches collaboratives

3.1 Utilisation du Leap Motion

Le Leap Motion a été assez compliqué à utiliser au début, car le nombre d'informations transmises par ses capteurs est plutôt importante : vitesse et position de chaque doigt, différenciation entre un doigt et un outil (un crayon par exemple), ainsi que le diamètre de la sphère virtuelle tangente au creux de la paume de la main (c'est la technique utilisée par le Leap pour mesurer l'ouverture de la main). Après avoir testé différentes démos exploitant la reconnaissance de tous les doigts, je me suis vite rendu compte que la plupart du temps le résultat en terme d'ergonomie était vraiment médiocre. En effet dans Unity, les forces appliqués à un objet lors d'une collision peuvent rapidement donner des résultats inexploitables si elles sont trop nombreuses. Or généralement quand tous les doigts de la main utilisés, chacun contrôle un objet distinct dans le jeu (une petite sphère par exemple). Et lorsque l'objet saisi se retrouve en collision avec 5 sphères appliquant des forces dans toutes les directions, il valse dans les airs !

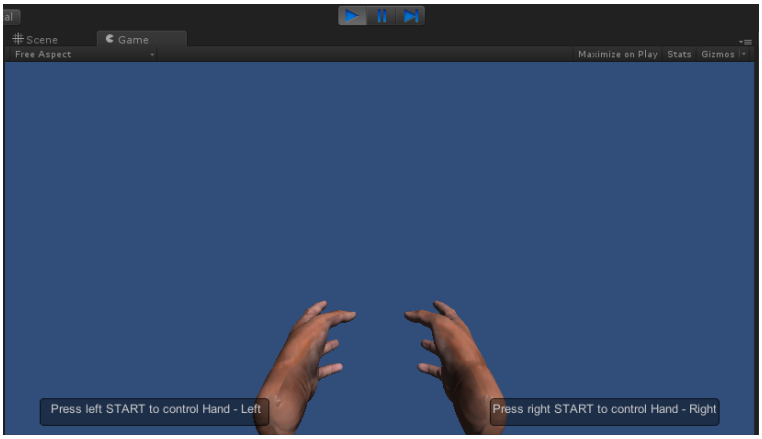
La méthode de contrôle retenue est donc plus simple : pour savoir si la main virtuelle doit saisir l'objet, on vérifie juste si elle est fermée, et on relâche l'objet si elle est ouverte (doigts visibles). Cette contrainte se traduit dans le programme par une variable booléenne beaucoup plus pratique à gérer.

Ensuite, la position d'une main dans l'espace permet d'avoir un contrôle sur trois axes : haut-bas, gauche-droite et avant-arrière. L'aspect ergonomique du programme consiste à attribuer ces axes à des actions qui vont paraitre logiques à l'utilisateur. Par exemple, l'axe gauche-droite correspondra à une rotation du bras robot sur le plan horizontal, tandis que l'axe haut-bas le dépliera pour qu'il puisse saisir des objets à différentes hauteurs. Les scripts correspondants sont visibles en annexe.

3.2 Utilisation du Razer Hydra

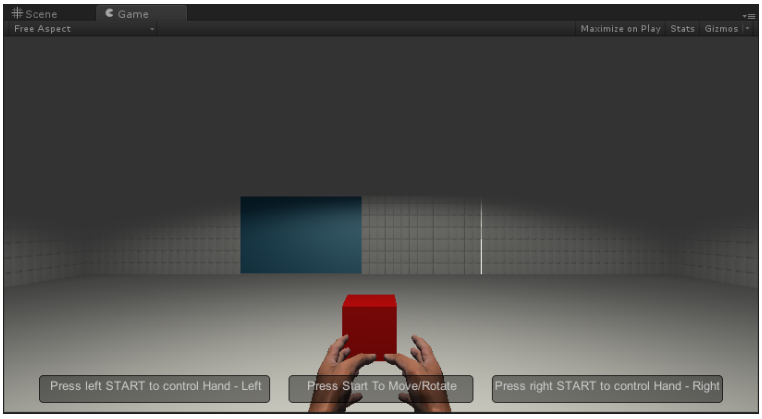
Pour utiliser le Razer Hydra simplement en tant que « manette », pour jouer directement, il suffit de télécharger le pilote correspondant sur le site de Razer (le pilote qui s'installe de lui-même étant relativement lent). Ce qui a été fait, étant donné que nous voulions nous en servir comme manette pendant les tests. Mais il fallait surtout que nous puissions développer notre application avec le Hydra sur Unity. Nous savions que l'Asset Store d'Unity contenait des outils d'interaction avec de nombreux périphériques, nous en avons donc cherché un pour le Razer Hydra, que nous avons trouvé : Sixense. Ce nom correspond en fait à l'entreprise qui possède maintenant le RH.

Donc, après avoir téléchargé le package Sixense et le driver, j'ai branché le RH pour le tester (sans Unity dans un premier temps). L'essai s'est avéré très concluant, on se retrouve avec un contrôleur qui permet de naviguer très efficacement dans Windows (comme une souris en fait). Mais il fallait ensuite tester ce périphérique avec Unity. Après avoir importé le package (dans un projet vide, pour ne pas avoir d'interactions non désirées), j'ai testé la « démo ».



La démo du Razer Hydra dans Unity

Celle-ci présente deux mains, chacune correspondant à une des manettes du RH.Au début, si le Razer Hydra est branché, il nous est demandé de pointer les manettes (« controllers» en anglais) vers le récepteur et d'appuyer sur les gachettes, afin de calibrer le système. Par la suite, un appui sur le bouton start (comme on le voit sur l'image ci-dessus) nous donne le contrôle d'une main. Après cet essai, j'étais donc convaincu que ce package était bien celui qu'il nous fallait pour exploiter pleinement le RH en tant qu'outil de développement. Je suis donc passé à l'étape suivante, à savoir la possibilité de se déplacer dans la scène pour l'utilisateur, ainsi que la gestion de la caméra. J'ai longtemps testé ces deux solutions séparement, en déplaçant la caméra en fonction des actions de l'utilisateur sur les joysticks des RH, et en faisant de même pour les mains. Je me suis alors rendu compte qu'il existait une solution bien plus simple, à savoir le parentage. C'est une technique connue en langage objet, qui consiste à désigner un objet comme « enfant » d'un autre, afin qu'il acquière ses caractéristiques .Sous Unity, la notion est légèrement différente, à savoir que l'enfant est dépendant des déplacements du parent, et qu'il n'en « dérive » pas. Il s'agit donc d'une bonne solution ici, car en désignant la caméra en tant qu'objet parent et les mains en tant qu'enfants, celles-ci suivront automatiquement le mouvement de la caméra, que ce soit en rotation (pour regarder à droite et à gauche en l'occurrence), ou en translation. Et cela est très intéressant, car peu importe quel objet les mains suivent, tant que ce dernier se comporte comme une caméra. Et ça sera un gros avantage lors de l'implémentation de l'Oculus Rift Pour ces tests, j'ai décidé de ne pas travailler directement dans Robots Lab, afin d'alléger le projet lors de l'exécution. J'ai donc crée une petite pièce, de type laboratoire également, inspirée par des jeux comme Portal.

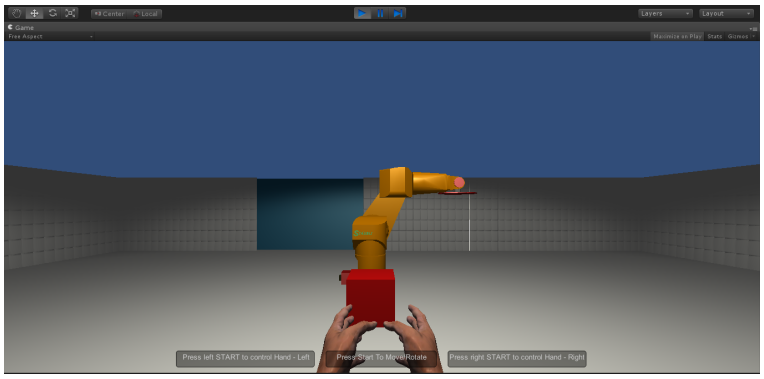


La salle de test

Le cube rouge que l'on voit sur l'image est utilisé pour les tests de préhension d'objets (que ce soit par l'utilisateur ou un robot).Pour la partie utilisateur, j'ai hésité quant à la manière de faire saisir l'objet par les mains. J'ai d'abord pensé à une saisie « réelle » utilisant des forces créées par les mains et forçant l'objet à rester « coincé » entre les deux mains. Mais je me suis vite rendu compte qu'il s'agissait d'une solution très difficile à réaliser, impliquant des calculs compliqués pour un résultat pas optimal.

J'ai donc décidé d'essayer les Raycast. Il s'agit d'un système très utilisé par les adeptes d'Unity. Il consiste à créer un rayon, qui s'il est coupé par un objet, renvoie la valeur « vrai », sur le principe d'un rayon infrarouge utilisé dans certains capteurs. On peut choisir le point de départ du rayon et sa direction. En cherchant des manières d'utiliser

le Leap Motion, nous avons trouvé sur l'Asset Store un exemple que j'ai décidé de transposer au RH, la seule différence étant la reconnaissance de la commande d'entrée. Il m'a donc suffit de remplacer l'entrée « Leap » par l'entrée « RH ».Sachant que le package Sixense donne une excellente description de tous les boutons, gâchettes et axes du RH, cela s'est avéré aisé. J'ai donc pu déplacer le cube via le Razer Hydra. Ceci est fait grâce au script SixenseHandController, qui dérive de SixenseObjectController. Lors de l'appui sur le bouton désiré, on déclenche l'animation préenregistrée correspondante, puis on crée le Raycast. Si celui-ci rencontre un objet, il le considère comme « carried_object », et lors de l'appel de la fonction MoveCarriedObject (qui se fait chaque update), si un objet est trouvé, celui-ci suivra le placement de la main. On peut galement dclencher les autres animations, grce l'appel d'un update consacr aux animations (ci sont assez proches du cube. Cependant, j'ai pas eu le temps de travailler ce point, ce qui a occasionn un manque de prcision. Pour teste



Le robot Staubli intégré à la scène

3.3 Utilisation de l'Oculus Rift

Pour augmenter le réalisme et l'immersion de notre scène, nous avons donc décidé d'inclure l'Oculus Rift. Après avoir téléchargé le package correspondant sur le site d'Oculus, nous l'avons inclus dans une scène toute simple, comprenant simplement le robot Staubli. Ce package comprend deux objets : un cameracontroller et un playercontroller, le playercontroller étant enfant de la caméra, afin de pouvoir suivre cette dernière. La caméra contient elle-même deux caméras, correspondant aux caméras de chaque il. Une fois ces objets importés et, point extrêmement important, toutes les autres caméras de la scène désactivées, on peut lancer le projet. On aperçoit bien sûr à l'écran l'image classique décrite précédemment, avec deux images. Mais une fois le casque enfilé, on peut tourner la tête et observer la scène. Après cela, nous avons inclus l'Oculus dans Robots Lab et nous avons pu profiter de notre scène au mieux.

Ayant travaillé séparément et souhaitant avoir un seul et unique projet, tout ce qu'il nous restait à faire était d'exporter nos parties de projets en tant que prefabs. Et d'importer ces derniers dans Robots Lab. Ceci fait, il suffisait de parenter les mains du Razer à la caméra comme je les avait parentées à la caméra principale lors de mes tests, afin qu'elles suivent ses mouvements, cette fois guidés par l'Oculus Rift. Afin de pouvoir opérer des interactions avec les robots, nous avons également importé ces derniers et placé le Staubli au centre de la pièce. Cependant, les interactions de celui-ci avec les objets ne s'opéraient pas correctement. Et ceci est un des axes d'amélioration possibles.

Test de l'implémentation de l'Oculus Rift au projet

Chapitre 4

Le ressenti de l'utilisateur

Pouvoir étudier les réactions et le ressenti des personnes expérimentant Virtual Robots est un des aspects importants de ce projet. A défaut d'avoir eu l'occasion de faire essayer l'application par différentes personnes, nous pouvons détailler ici quels sont les éléments intéressants à observer pour mesurer l'impact qu'a eu le jeu sur l'utilisateur.

4.1 Paramètres à prendre en compte

Les données à analyser se divisent en deux catégories : objectives et subjectives. Les données objectives sont celles qui peuvent être directement extraites de la session de jeu, par exemple le temps passé par type d'activité et par robot, ainsi que des informations d'ordre physiologique (battements de cur, transpiration, etc) qui peuvent être récoltées si le joueur est équipé des capteurs adéquats. Quant aux données subjectives, elles correspondent au ressenti de l'utilisateur, son avis et ses impressions sur ce qu'il à vécu.

4.1.1 Données objectives

Pour tirer parti des données objectives, il faudrait rajouter un script permettant d'enregistrer à intervalles réguliers toutes les informations requises, pour ensuite les enregistrer dans un fichier CSV par exemple, ce qui permettrait ensuite d'exporter ces données vers une autre application pour créer des infographies synthétiques.

4.1.2 Données subjectives

Le meilleur moyen de récupérer ces informations est tout simplement de les demander aux personnes ayant testé l'application, au moyen d'un questionnaire. Un exemple de questions pertinentes est présenté ci-dessous :

1. Quel robot vous a semblé le plus amusant ?
2. Quel robot vous a semblé le plus polyvalent ?
3. Quelle activité avez-vous préféré ? Pourquoi ?
4. Aviez-vous l'impression de jouer contre une machine ou contre un être doué de conscience ? (question à poser pour chaque robot)
5. Quelle a été votre sentiment principal lors des échanges avec le robot ? (question à poser pour chaque robot)
6. Auriez-vous envie d'interagir avec des robots au quotidien ? Pourquoi ?
7. Pensez-vous qu'il soit possible d'avoir un comportement social avec un robot ?

4.2 Interprétation des données

Pour interpréter correctement toutes ces informations et en tirer des conclusions pertinentes, cela nécessite des connaissances en psychologie et en sociologie. Nous n'avons malheureusement pas ces connaissances, mais il serait très intéressant de faire appel à des étudiants des facultés correspondantes afin d'enrichir ce projet.

Chapitre 5

Conclusion

5.1 Apports personnels

Adrien : Ce projet m'a permit de développer mes connaissances dans deux domaines qui me tiennent à cur, la réalité virtuelle et la robotique. En effet, j'ai pu m'entrainer à l'utilisation d'Unity 3D, qui est un des moteurs de jeu les plus utilisés en réalité virtuelle. J'ai également eu l'occasion d'apprendre à utiliser le Leap Motion, qui est un périphérique prometteur et se montre vraiment intéressant à utiliser pour certaines taches. Pour la partie robotique, j'ai apprécié le fait de transposer des mécanismes réels en virtuel, car cela permet de bien comprendre comment les différentes pièces du robot sont articulées entre elles. Au final, programmer les mouvements d'un robot virtuel est assez similaire au fait de le faire en vrai, à la différence qu'ici un simple ordinateur suffit.

Julien : Ce projet m'a été utile sur trois points particuliers : le premier, qui est je suppose commun à un grand nombre de projets, est la gestion de celui-ci, à savoir comment se lancer, comment organiser son temps et ses ressources en fonction de ce qu'on à faire. Et aussi comment se répartir le travail, afin que chacun puisse également se servir des travaux de l'autre. Le second point concerne la réalité virtuelle et les IHM, étant donné que nous avons utilisé les principales interfaces connues, mis à part tout ce qui concerne les casques de contrôle par la pensée (chose que j'avais déjà utilisée lors de mon stage EI3). Ainsi, cela donne une bonne idée du paysage actuel de la RV et donne un avantage certain quant à une candidature éventuelle. De même, utiliser Unity 3D pour des choses plus compliquées que celles vues en cours donne également ce genre d'avantage.

5.2 Axes d'amélioration

Les autres axes concernent principalement le nombre d'interactions possibles avec chaque robot et la précision de ces dernières. Ayant travaillé sur les activités du Staubli, j'ai surtout eu une idée concernant le jeu de Nim qu'il est possible de mettre en place avec ce robot et des cubes à la place des classiques allumettes. Et il serait aisé d'ajouter une IA au robot, grâce à l'algorithme minmax , très utile et pertinent dans ce genre de cas. Ainsi, plus la profondeur de l'arbre est importante, plus l'utilisateur aura l'impression d'interagir avec un humain, ce qui est évidemment l'effet que nous recherchons.

5.3 Abstract

Today robots are limited to industrial purposes, but public models begin to spread around the world. To help the studying of the new social interaction which will be developed between human and robots in the near future, it is useful to incorporate elements of virtual reality. This allows simulating the interactions between a person of the physical world and a virtual robot. Several activities are possible for the user : stacking cubes with the help of the robot, exchanging objects, miming the movements The virtual world of this social experiment was created with the videogame engine Unity3D, and some of the assets were modeled in 3 dimensions using the popular free tool Blender. To give the user a natural and intuitive feeling when interacting with the robots, different innovative human-machine interfacing devices are supported : the Leap Motion which can track the hands movements without touching anything, and the Razer Hydra which is composed of two controllers able to detect the hands position in space. The display is made through the Oculus Rift, a 3D screen fitted into a head-mounted device. On the virtual side, two very different robots are present : an industrial arm and the humanoid shaped Nao. The point of

choosing different input peripherals and robots is to study users' reactions according to the situation, and so to determine what configuration gives the most immersive feeling. The 3D environment is completely implemented and all devices work properly. Now the only task remaining is to test the system under real conditions and collect the results.

Keywords :
- virtual reality
- robots
- human-machine interfaces
- feelings
- social

Chapitre 6

Annexes

6.1 Scripts pour le Leap Motion

Ce premier script sert à se déplacer en vue à la première personne dans une scène, et de déplacer des objets en fermant le poing lorsque l'on se trouve face à eux (On notera pour cela l'utilisation d'un raycast ligne 41-44).

```
1  using UnityEngine;
2  using System.Collections;
3  using Leap;
4
5  public class LeapCharacterController : MonoBehaviour {
6
7    Controller m_leapController;
8    float m_lastBlastTime = 0.0f;
9
10   GameObject m_carriedObject;
11   bool m_handOpenThisFrame = false;
12   bool m_handOpenLastFrame = false;
13
14   // Use this for initialization
15   void Start () {
16     m_leapController = new Controller();
17   }
18
19   // gets the hand furthest away from the user (closest to the screen).
20   Hand GetForeMostHand() {
21     Frame f = m_leapController.Frame();
22     Hand foremostHand = null;
23     float zMax = -float.MaxValue;
24     for(int i = 0; i < f.Hands.Count; ++i) {
25       float palmZ = f.Hands[i].PalmPosition.ToUnityScaled().z;
26       if (palmZ > zMax) {
27         zMax = palmZ;
28         foremostHand = f.Hands[i];
29       }
30     }
31
32     return foremostHand;
33   }
34
35   void OnHandOpen(Hand h) {
36     m_carriedObject = null;
37   }
38
39   void OnHandClose(Hand h) {
40     // look for an object to pick up.
41     RaycastHit hit;
42     if(Physics.SphereCast(new Ray(transform.position + transform.forward * 2.0f, transform.forward
           ), 2.0f, out hit)) {
43       m_carriedObject = hit.collider.gameObject;
44     }
45   }
46
```

```
47    bool IsHandOpen(Hand h) {
48      return h.Fingers.Count > 1;
49    }
50
51    // processes character camera look based on hand position.
52    void ProcessLook(Hand hand) {
53      float rotThreshold = 0.0f;
54      float handX = hand.PalmPosition.ToUnityScaled().x;
55
56      if (Mathf.Abs(handX) > rotThreshold) {
57        Camera.main.transform.RotateAround(Vector3.up, handX * 0.03f);
58      }
59    }
60
61    void MoveCharacter(Hand hand) {
62      if (hand.PalmPosition.ToUnityScaled().z > 0) {
63        Camera.main.transform.position += Camera.main.transform.forward * 0.1f;
64      }
65
66      if (hand.PalmPosition.ToUnityScaled().z < -1.0f) {
67        Camera.main.transform.position -= Camera.main.transform.forward * 0.04f;
68      }
69    }
70
71    // Determines if any of the hand open/close functions should be called.
72    void HandCallbacks(Hand h) {
73      if (m_handOpenThisFrame && m_handOpenLastFrame == false) {
74        OnHandOpen(h);
75      }
76
77      if (m_handOpenThisFrame == false && m_handOpenLastFrame == true) {
78        OnHandClose(h);
79      }
80    }
81
82    // if we're carrying an object, perform the logic needed to move the object
83    // with us as we walk (or pull it toward us if it's far away).
84    void MoveCarriedObject() {
85      if (m_carriedObject != null) {
86        Vector3 targetPos = transform.position + new Vector3(transform.forward.x, 0, transform.
              forward.z) * 5.0f;
87        Vector3 deltaVec = targetPos - m_carriedObject.transform.position;
88        if (deltaVec.magnitude > 0.1f) {
89          m_carriedObject.rigidbody.velocity = (deltaVec) * 10.0f;
90        } else {
91          m_carriedObject.rigidbody.velocity = Vector3.zero;
92        }
93      }
94    }
95
96    void FixedUpdate () {
97      Hand foremostHand = GetForeMostHand();
98      if (foremostHand != null) {
99        m_handOpenThisFrame = IsHandOpen(foremostHand);
100       ProcessLook(foremostHand);
101       MoveCharacter(foremostHand);
102       HandCallbacks(foremostHand);
103       MoveCarriedObject();
104     }
105     m_handOpenLastFrame = m_handOpenThisFrame;
106   }
107 }
```

Listing 6.1 – Leap Character Controller.cs

Le script ci-dessous permet de contrôler directement le bras robot Staubli à l'aide d'une seule main. La position de la main gère la rotation et le dépliement du bras (ligne 31-34), tandis que l'ouverture/fermeture de la main actionne la pince (ligne ligne 36-48). Ces tests sont effectués uniquement si une main est détectée (ligne 27-29).

```
1  using UnityEngine;
2  using System.Collections;
3  using Leap;
```

```
4
5  public class leap_control : MonoBehaviour {
6    public GameObject[] fingers;
7    public GameObject[] colliders;
8
9    private LeapManager _leapManager;
10   // Use this for initialization
11
12
13   public Transform servo1;
14   public Transform servo2;
15   public Transform servo3;
16   public Transform left_finger;
17   public Transform right_finger;
18
19   bool close=false;
20
21   void Start () {
22     _leapManager = (GameObject.Find("LeapManager")as GameObject).GetComponent(typeof(LeapManager))
           as LeapManager;
23   }
24
25   // Update is called once per frame
26   void Update () {
27     Hand primeHand = _leapManager.frontmostHand();
28
29     if(primeHand.IsValid)
30     {
31       Vector3[] fingerPositions = _leapManager.getWorldFingerPositions(primeHand);
32       servo1.transform.Rotate(0,primeHand.PalmPosition.ToUnityTranslated().x/5,0);
33       servo2.transform.Rotate(-primeHand.PalmPosition.ToUnityTranslated().y/10,0,0);
34       servo3.transform.Rotate(primeHand.PalmPosition.ToUnityTranslated().y/10,0,0);
35
36       if(LeapManager.isHandOpen(_leapManager.frontmostHand())&&close)
37       {
38         left_finger.transform.Rotate(0,30,0);
39         right_finger.transform.Rotate(0,-30,0);
40         close=false;
41       }
42
43       if(!LeapManager.isHandOpen(_leapManager.frontmostHand())&&!close)
44       {
45         left_finger.transform.Rotate(0,-30,0);
46         right_finger.transform.Rotate(0,30,0);
47         close=true;
48       }
49     Debug.Log(primeHand.PalmPosition.ToUnityTranslated());
50   }
51 }
52 }
```

Listing 6.2 – leap$_c$*ontrol.cs*

6.2 Scripts pour le Razer Hydra

```
1  //
2  // Copyright (C) 2013 Sixense Entertainment Inc.
3  // All Rights Reserved
4  //
5
6  using UnityEngine;
7  using System.Collections;
8
9  public class SixenseHandController : SixenseObjectController
10 {
11   protected Animator      m_animator = null;
12   protected float       m_fLastTriggerVal = 0.0f;
13   //public GameObject Cube ;
14   GameObject m_carriedObject;
```

```
15  GameObject target;
16  Vector3 dir;
17  bool left=false;
18  bool right=false;
19
20  protected override void Start()
21  {
22    // get the Animator
23    m_animator = this.gameObject.GetComponent<Animator>();
24
25    base.Start();
26  }
27
28  protected override void UpdateObject( SixenseInput.Controller controller )
29  {
30    if ( m_animator == null )
31    {
32      return;
33    }
34
35    if ( controller.Enabled )
36    {
37      // Animation update
38      UpdateAnimationInput( controller );
39
40    }
41
42    base.UpdateObject(controller);
43  }
44
45
46  void OnGUI()
47  {
48    if ( Hand == SixenseHands.UNKNOWN )
49    {
50      return;
51    }
52
53    if ( !m_enabled )
54    {
55      int labelWidth = 250;
56      int labelPadding = 120;
57      int horizOffset = Hand == SixenseHands.LEFT ? -labelWidth - labelPadding  : labelPadding;
58
59      string handStr = Hand == SixenseHands.LEFT ? "left" : "right";
60      GUI.Box( new Rect( Screen.width / 2 + horizOffset, Screen.height - 40, labelWidth, 30 ),  "
            Press " + handStr + " START to control " + gameObject.name );
61    }
62  }
63
64
65  void MoveCarriedObject() {
66    if (m_carriedObject != null) {
67      Vector3 targetPos = transform.position + new Vector3(transform.forward.x, 0, transform.
            forward.z) * 5.0f;
68      Vector3 deltaVec = targetPos - m_carriedObject.transform.position;
69      if (deltaVec.magnitude > 0.1f)
70      {
71        m_carriedObject.rigidbody.velocity = (deltaVec) * 10.0f;
72      }
73
74      else
75      {
76        m_carriedObject.rigidbody.velocity = Vector3.zero;
77      }
78    }
79  }
80
81  void FixedUpdate () {
82      MoveCarriedObject();
83
```

```
84      }
85
86    // Updates the animated object from controller input.
87    protected void UpdateAnimationInput( SixenseInput.Controller controller)
88    {
89      // Point
90      if ( Hand == SixenseHands.RIGHT ? controller.GetButton(SixenseButtons.ONE) : controller.
            GetButton(SixenseButtons.TWO) )
91      {
92        m_animator.SetBool( "Point", true );
93      }
94      else
95      {
96        m_animator.SetBool( "Point", false );
97      }
98
99      // Grip Ball
100     float fTriggerVal = controller.Trigger;
101     fTriggerVal = Mathf.Lerp( m_fLastTriggerVal, fTriggerVal, 0.1f );
102     m_fLastTriggerVal = fTriggerVal;
103     if ( fTriggerVal > 0.01f )
104     {
105       m_animator.SetBool( "GripBall", true );
106       RaycastHit hit;
107       if(Physics.SphereCast(new Ray(transform.position + transform.forward * 2.0f, transform.
              forward), 2.0f, out hit)) {
108       m_carriedObject = hit.collider.gameObject;
109     }
110
111     }
112     else
113     {
114       m_animator.SetBool( "GripBall", false );
115       m_carriedObject = null;
116     }
117
118
119     // Hold Book
120     if ( Hand == SixenseHands.RIGHT ? controller.GetButton(SixenseButtons.TWO) : controller.
            GetButton(SixenseButtons.ONE)  )
121     {
122       m_animator.SetBool( "HoldBook", true );
123
124     }
125     else
126     {
127       m_animator.SetBool( "HoldBook", false );
128     }
129
130     // Fist
131     if ( Hand == SixenseHands.RIGHT ? controller.GetButton(SixenseButtons.THREE) : controller.
            GetButton(SixenseButtons.FOUR) )
132     {
133       m_animator.SetBool( "Fist", true );
134
135     }
136     else
137     {
138       m_animator.SetBool( "Fist", false );
139     }
140
141
142
143     // Idle
144     if ( m_animator.GetBool("Fist") == false &&
145        m_animator.GetBool("HoldBook") == false &&
146        m_animator.GetBool("GripBall") == false &&
147        m_animator.GetBool("Point") == false )
148     {
149       m_animator.SetBool("Idle", true);
150     }
```

```
    else
    {
      m_animator.SetBool("Idle", false);
    }
  }
}
```

Listing 6.3 – SixenseHandController.cs

```
using UnityEngine;
using System.Collections;

public class SixenseObjectController : MonoBehaviour {

  public SixenseHands     Hand;
  public GameObject       camera;
  public GameObject       LeftHand;
  public GameObject     RightHand;
  public Vector3        Sensitivity = new Vector3( 0.01f, 0.01f, 0.01f );

  protected bool        m_enabled = false;
  protected Quaternion    m_initialRotation;
  protected Vector3     m_initialPosition;
  protected Vector3     m_baseControllerPosition;

  // Use this for initialization
  protected virtual void Start()
  {
    m_initialRotation = this.gameObject.transform.localRotation;
    m_initialPosition = this.gameObject.transform.localPosition;
  }

  // Update is called once per frame
  void Update ()
  {
    if ( Hand == SixenseHands.UNKNOWN )
    {
      return;
    }

    SixenseInput.Controller controller = SixenseInput.GetController( Hand );
    if ( controller != null && controller.Enabled )
    {
      UpdateObject(controller);
    }
  }


  void OnGUI()
  {
    if ( !m_enabled )
    {
      GUI.Box( new Rect( Screen.width / 2 - 100, Screen.height - 40, 200, 30 ),  "Press Start To
          Move/Rotate" );
    }
  }


  protected virtual void UpdateObject(  SixenseInput.Controller controller )
  {
    if ( controller.GetButtonDown( SixenseButtons.START ) )
    {
      // enable position and orientation control
      m_enabled = !m_enabled;

      // delta controller position is relative to this point
      m_baseControllerPosition = new Vector3( controller.Position.x * Sensitivity.x,
                          controller.Position.y * Sensitivity.y,
                          controller.Position.z * Sensitivity.z );

      // this is the new start position
```

```
62        m_initialPosition = this.gameObject.transform.localPosition;
63      }
64
65      if ( m_enabled )
66      {
67        UpdatePosition( controller );
68        UpdateRotation( controller );
69      }
70
71      if ( controller.Enabled )
72      {
73
74        if(Hand == SixenseHands.LEFT)
75        {
76          camera.transform.Translate (controller.JoystickX * Time.deltaTime*3,0,controller.JoystickY
                * Time.deltaTime*3);
77          //EmptyLeftHand.transform.Translate(0,controller.JoystickY * Time.deltaTime,0);
78          //EmptyRightHand.transform.Translate(0,controller.JoystickY * Time.deltaTime,0);
79        }
80
81        if(Hand == SixenseHands.RIGHT)
82        {
83          camera.transform.Rotate(0,controller.JoystickX*Time.deltaTime*40,0);
84          //EmptyLeftHand.transform.Rotate(0,controller.JoystickX*Time.deltaTime*40,controller.
                JoystickY*Time.deltaTime*40);
85          //EmptyRightHand.transform.Rotate(0,controller.JoystickX*Time.deltaTime*40,controller.
                JoystickY*Time.deltaTime*40);
86        }
87
88      }
89
90
91    }
92
93
94    protected void UpdatePosition( SixenseInput.Controller controller )
95    {
96      Vector3 controllerPosition = new Vector3( controller.Position.x * Sensitivity.x,
97                            controller.Position.y * Sensitivity.y,
98                            controller.Position.z * Sensitivity.z );
99
100     // distance controller has moved since enabling positional control
101     Vector3 vDeltaControllerPos = controllerPosition - m_baseControllerPosition;
102
103     // update the localposition of the object
104     this.gameObject.transform.localPosition = m_initialPosition + vDeltaControllerPos;
105   }
106
107
108   protected void UpdateRotation( SixenseInput.Controller controller )
109   {
110     this.gameObject.transform.localRotation = controller.Rotation * m_initialRotation;
111   }
112 }
```

Listing 6.4 – SixenseObjectController.cs

```
1  using UnityEngine;
2  using System.Collections;
3
4  public class RaycastControl : MonoBehaviour {
5
6    GameObject m_carriedObject;
7    public GameObject Cube;
8    public GameObject magnet;
9
10
11   // Use this for initialization
12   void Start () {
13
14   }
```

```
15
16    // Update is called once per frame
17    void Update () {
18
19      Debug.Log(Cube.transform.position.y+"/"+this.transform.position.z);
20      Debug.Log(Cube.transform.position.z+"/"+this.transform.position.x);
21
22
23      if ((Cube.transform.position.y == this.transform.position.z) && (Cube.transform.position.z ==
            this.transform.position.x))
24      {
25
26        Debug.Log("catch");
27        Cube.transform.parent = this.transform;
28      }
29
30    }
31  }
```

Listing 6.5 – RaycastControl.cs