Immersive interaction framework for self-reconfigurable modular robots

---

Semester project
fall 2015

Valentin NIGOLIAN
valentin.nigolian@epfl.ch

Biorobotics Laboratory EPFL
Professor : Auke J. Ijspeert
Supervisors : Mehmet Mutlu, Stéphane Bonardi and Simon Hauser

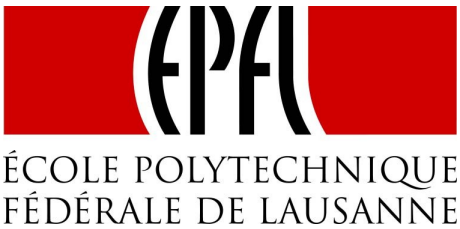

Table of contents

Abstract

Roombots are self-reconfigurable modular robots. They are one of the current projects of BioRob, the Biorobotics Laboratory at EPFL. Its main purpose is to create self-reconfigurable adaptive furniture and has many applications. One of them is to make disabled people more independent by providing a smart interface adapted to their daily physical challenges, allowing them to move their furniture by themselves through the use of Roombots. This project explores new ways to interact with modular robots by using a gesture-based interface with a Virtual Reality (VR) visual feed-back, for a more immersive apprehension of the virtual world. The goal is to enable the user to quicly set up a room using furniture made out of Roombots modules and then finally visualize the Roombots building the desired furnitures.

Acknowledgements

I would like to thank my supervisors, Mehmet Mutlu, Stéphane Bonardi and Simon Hauser for their precious advice and guidance during the whole project, along with Professor Auke Ijspeert who gave me the oportunity to work on such an interesting project. I have learned a lot and working with such promising and various technologies gave me nothing but a taste for scientific research and exploration. I wish all my future project will be as intellectually rewarding.

I would also like to thank the members of BioRob who took some of their time to test the software, along with my friends Jonathan and Laurette and my father and my brother Keyan who also tested the software. Finally, I would like to thank Simon who took the nice pictures you can see in the present document.

1. Motivation

1.1 Roombots

Roombots are self-reconfigurable modular robots consisting of two sphere-like objects bound together that have three rotation axes, giving every module three degrees of freedom and allowing them to turn in every direction[1]. Furthermore, each module has both active and passive connectors that let them attach to each other in order to build dynamic and more complex structures. It also makes them able to work as locomotion devices, giving them freedom to move around in space. If the environment provides dedicated passive grids, they can also climb walls and even ceilings. Possibilities of the Roombot modules are covered in multiple articles from BioRob[2] [3] [4] [5] [6].
This project adds to the whole Roombots project in that it gives a new way to interact with them to set goal positions for self-reconfiguration tasks.

1.2 Previous work

Over the last few years, various interfaces to control the Roombots were designed at BioRob. It started in 2010 with the work of G. Gressier[7] who designed an interface to visualize the rotations and positioning of the Roombots module.
Then, a more mobile-oriented approach was taken in the 2011 work of J. Blatter[8] who designed an iPad application using augmented reality to add furniture into a scene captured by the iPad's camera. In his work, we can find a first attempt to visualize a whole scene containing furnitures made of Roombots. A second work in this augmented reality approach was done in 2014 by L. Girod.[9] Gesture-recognition was then integrated in the whole Roombots project with a 2014 project by A. Ozgür[10], in which he used Kinect cameras to track his body, allowing him to point positions on a grid with his arm and making a Roombots module go to that position.
Finaly, a rather bold input device was tested by M. Moret in his 2015 work[11], in which he attempted to control Roombots with a Brain Computer Interface (BCI) using Electroencephalography (EEG).
All those previous projects lead to the decision to take a step further and combine gesture-recognition with Virtual Reality and the beginning of the project presented in this document. Indeed, while all previous project (except for M. Moret's project) explored more classic approaches regarding the interaction with robots, the present project explores the use of the latest commercialy available technologies in the virtual interaction domain.

1.3 Goals

The main and ultimate goal of this project is to provide an immersive interaction framework for modular robots, in particular Roombots. It means that it would allow a user to immerse him or herself into a virtual environment containing furnitures made of Roombots and then interact with these structures with gesture-recognition. By using the software developed throughout this project, one could set up a complete room, and then visualize how Roombots modules would behave and move when building this room. Its main application would be to give people the power to change their room, without the need to move real furnitures around. This would be even more interesting for disabled people since it would make it more than practical, but possible to change their room by themself, ultimately making them more

independent. To simulate the behaviour of the modules, we would use a various range of path-finding and reconfiguration algorithms, with the possibility to test them out by getting a direct feed-back of the use of a particular algorithm in Virtual Reality, making it the second application of this project. The initial goals of the project were to first create a virtual environment using the Oculus Rift and then integrate the Million Modules March algorithm, a well-known reconfiguration algorithm designed by R. Fitch and Z. Butler[12] and explored further by R. Fitch and R. McAllister in a more recent article[13]. The first part, namely creating a virtual environment using the Oculus Rift that would include Roombots-made structures was divided into the following steps :

- 3D scene rendering
Integrating an OpenGL[14] context and modeling a simple room, creating a visual context for our Roombots-made structures and interface.

- Scene rendering through the Oculus Rift
Once the scene is created, rendering it through the Oculus Rift to get a more immersive feedback. Implementing a more classic representation, called the « mirror screen », allowing to also see the scene on a regular screen mirroring the image displayed on the Rift.

- Roombots and structures implementation and development of a classic graphic interface.
Integrating simple robots and structures models and implementing a classic interface using the mouse and the keyboard allowing the user to add, move or remove said structures.

The initial plan also accounted for a few upgrades, the most important one being the integration of gesture recognition and the upgrading of the interface to use said gesture recognition. But as weeks went by, the load of work needed to perform the first assignment appeared to be too heavy to let enough space for the implementation of the cited algorithm. In the end, it was decided to limit ourself to a much simpler algorithm, but to design the software in such a way that it would be possible to implement new algorithms. This aspect will be detailed later, when we will be talking about the software itself in section 3.5. Furthermore, the classic interface was abandoned for practical reasons, since it would have taken the same amount of time as to develop the gesture-based interface.Again, this will be explored in more details in section 3.3.

2. Hardware set-up

2.1 Immersive visual feed-back : The Oculus Rift

The Oculus Rift[15] is a Virtual Reality device consisting of a Head-Mounted Display (HMD) and an infrared camera (see fig 2.1). It can detect orientation and position to follow the user's head movements, giving the impression of actualy being in the virtual environment displayed on the HMD. For this project, we used the second Development Kit (DK2).
The Oculus Rift's Software Development Kit (SDK) is relatively easy to use and provides a few demos greatly helping developers to integrate the device's functionalities into any OpenGL-based or DirectX-based context. The SDK provides powerful ways to directly interact with all the hardware's components and only a few function calls is enough to retrieve the rotation and projection matrices corresponding to the HMD's orientation and characteristics (such as its Field Of View (FOV)), two upmost important mathematics objects when dealing with computer graphics. It is not invasive for the user since the HMD is cennected to the computer via only a single cable (that splits into a USB cable and a HDMI cable). Finaly, it is highly precise and reactive, as it follows all head movements almost instantly.

Fig 2.1 : The HMD on the left, the infrared camera on the rigt. The little red dots on the HMD are infrared lights invisible to the naked eye but captured by the camera

2.2 Immersive interaction : The Leapmotion

The Leapmotion[16] is a gesture-recognition device that works with infrared lights. It consists of a small USB-linked device that must be put on a table in front of the user (see fig 2.2 for a look at the device). For more information about the Leapmotion, please refer to the Leapmotion's official website[17]. The Leapmotion has a Field Of View (FOV) of 150° and a range going from 25mm to 600mm. It is spatially limited, but as long as the user is keeping his or her hands within its limitation, it is efficient in recognizing gestures. Indeed, the device already has built-in detection of a lot of typical hand gestures, such as pushing, circling, swiping or pinching, and its SDK makes it easy to use. For those reasons, it was chosen over other gesture-recognition devices like Microsoft's Kinect. Plus, the latter was already used with Roombots before in A. Ozgür's project cited in section 1.2 and I figured it would really add something new to use a different device. The Leapmotion can also detect when a user is using a pen or any other oblong-shaped object but no interest was found in this feature for the present project. Finally, it can also be attached to the HMD using a small adapter in order to track the hands relatively to the user's position and orientation rather than relatively to the table. But this would have made the interface more complex and after testing it, it appeared to be less reliable when tracking hand gestures and was thus not considered as a viable option.

Fig 2.2 : The Leapmotion device with its USB cable. Again, the red dots are infrared lights captured by the camera, but invisible to the naked eye.

2.3 The complete set-up

Tying everything together, we end up with a configuration where the user is facing the computer screen on which the Oculus's camera is placed. The operator uses the right hand above the Leapmotion to interact with the interface and the left hand on the keyboard while wearing the Oculus's HMD on the head. The Leapmotion provides gesture-recognition and complements the VR by giving the possibility to interact with the virtual environment simply by moving the hand and pinching structures with it. Also, the keyboard complements the Leapmotion in that it lets the user move freely in the room without having to physicaly move, something the Leapmotion could technicaly do but it would be counter-intuitive to control leg-related movements with hands. Finaly, the computer acts as the central unit tying all elements together.
Fig. 2.3 shows a typical use of the software with the full equipment.

Fig 2.3 : The complete set-up includes the Oculus Rift, the HMD and the infrared camera (in yellow), the Leapmotion device tracking the right hand above it (in red) and the left hand using the keyboard (in green)

3. Software development

We will now dwelve further the development process I followed through this project.
The full documentation of the code and its manual can be found in the git repository of the project[18].

3.1 Virtual Environment

The first goal was to set-up a simple virtual environment . To do so, the first thing was to explore the options regarding the various libraries for windows and OpenGL context, while trying to determine how the system would be designed before adding any use of the Oculus Rift. In the end, I chose freeGLUT[19] (free openGL Utility Toolkit) for the windowing system as it is very simple to use and allows a better control over the rendering loop than GLUT. For the OpenGL context manipulation, I chose GLEW[20] (openGL Extension Wrangler library), as it provides nice access to the most useful OpenGL extensions. For the matrix transformations and manipulations (an important aspect when dealing with graphics), I chose GLM[21] (OpenGL Mathematics) for its nice and simple functions and classes. Then, the main goal was to set up a dynamic OpenGL environment consisting of a simple quad for the floor and a large cube that would be used to display a sky (to get a nice view to render in the Rift later). The main work was to write all the code allowing the use of shaders and other OpenGL elements such as Vertex Array Objects or Vertex Buffer Objects. A ShaderLoader class was written to make the use of shaders much easier. This first step of coding was very successful, as the system permitted the creation and rendering of basic models and was ready to be upgraded to the next level. Then, textures were integrated into the system. A nice skybox texture was bound to the large cube and a generic floor texture for the quad. An already-existing graphics engine such as Ogre3D[22] could have been used but it was a choice to start from scratch to ensure total control over the whole software.

3.2 Using the Oculus Rift

Once the virtual environment was set-up, the next step was to use the Oculus Rift to render the scene. To do so, the OVR (Oculus Virtual Reality) API's documentation[23] and the Oculus' developer website were of great help. The first challenge was to make the camera inside the virtual world follow the HMD's orientation. That goal was achieved quite easily and the camera followed the HMD's movements smoothly. But the scene was still to be displayed on the Rift and it was tricky to say the least. Indeed, it implies, among other things, stereo rendering (one rendering for each eye), frame buffering, texture generation and mirror buffering (as the image is mirrored on the regular screen). In order to get a good rendering without spending days trying to solve problems that others already had solved, some code was taken from a demo delivered with the SDK (OculusRoomTiny (GL)) and integrated it into the code, with a lot of adaptation to fit the system. This code is under the Apache License 2.0[24] which allows such use of the code and therefore should not represent any problem. After having integrated the solution given in the SDK's example, the Rift was working perfectly fine and allowed the exact desired behavior regarding the control of the camera.

At this point, a new class called « RiftHandler » that is free from the rest of the system and makes the use of the Oculus Rift very simple was created. The main goal of such encapsulation was to make it really easy to re-use in another project using the Oculus Rift. For more details about how to use this code, please refer to the software's documentation. It should be noted that as much as this class is simple to use, its drawback is it also gives no precise control over the hardware and has a few limits. For example, it does not take the HMD's position into account, only its orientation. That means the moving around with the HMD does not displace the camera within the scene. Fig 3.2 shows the interface-free virtual environment as it displayed on the rift. Note that all screenshots from the software will be mirror images of what is displayed on the Rift and thus will always show the same image twice, once for each eye.

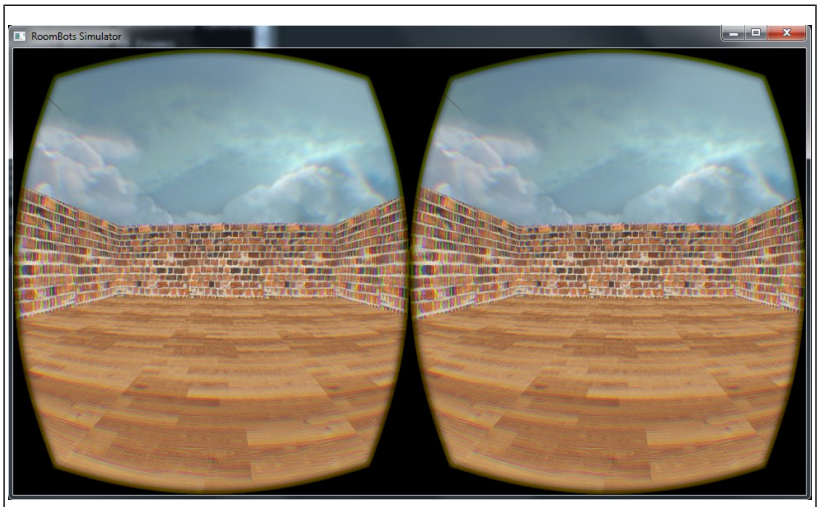

Fig 3.2 : The virtual environment without the GUI

3.3 Gesture-based GUI

Now that we could see the environment through the Oculus Rift, it was time to interact with it. The first step to do so was to make use of the Leapmotion device. As explained in the device's description, its SDK makes it really easy to include to any project. A few function calls suffice to get the position of any hand or finger, or other information like the palm's normal orientation. For this reason, it was pretty quick to integrate it into the project and after a short reflexion, it felt like the most natural way to interact with structures was to grab or « pinch » them and then to drop them whenever we wanted. This lead to a Graphic User Interface (GUI) consisting of « buttons », containers where structures would pop and a single pointer representing the hand and following its movements. Both the buttons and the pointer are represented as cubes, blue and red respectively (see fig. 3.3.1 for a look at the GUI with the buttons and pointer). At first, before starting to build Roombots-made structures, the system drew structures as green cubes in order to put the emphasis on how to make the interface as intuitive as possible before spending time to make it look better.

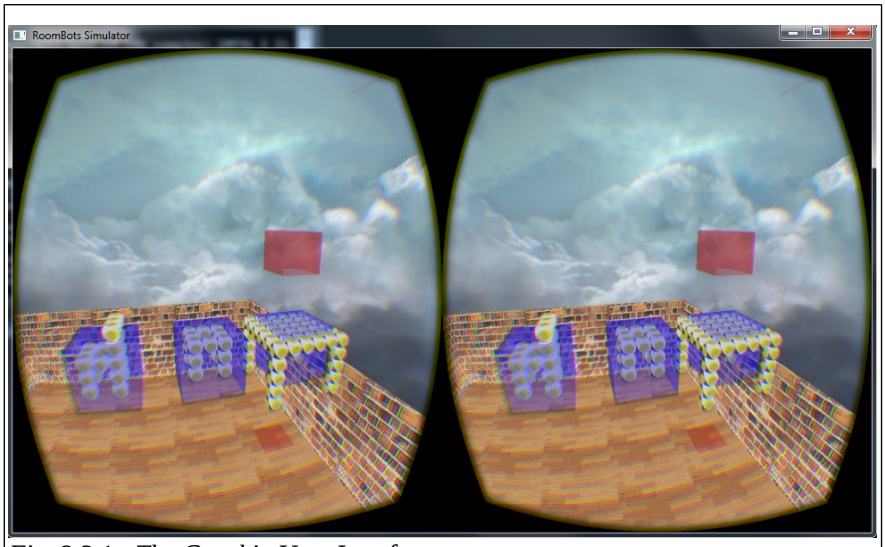

Fig. 3.3.1 : The Graphic User Interface

The Leapmotion has its own coordinate system[25] and it required some effort to translate the raw data obtained from the device into the software coordinate system. After some lengthy sessions of fine-tuning and tweaking of the parameters of the Leapmotion, the interface allowed to indeed grab and drop structures, with new structures popping everytime one is dropped. After that, Roombots were integrated as 3D models into the scene through the importation of .obj (wavefront) files[26], a standard format for 3D meshes (models). The source file came from previous works using 3D representation of Roombots. With this Roombots model, it became possible to design complex structures such as chairs or tables. Those were imported from a new file format specifically designed for this project, .rbs files (RoomBot Structure). It is a very simple file format consisting of a series of triplets of integers where each pair of triplet represents a Roombot module as the positions of its two half-modules. .rbs files are imported into the system and used to create beautiful Roombots-made structures into the scene. This latter element made it possible to use the creations of R. P. Dryzner, a student doing another project with Roombots, where structures are created from play-doh. He implemented a feature exporting his structures into said .rbs files, building a bridge between our two projects (see fig. 3.3.2 for examples of his structures).

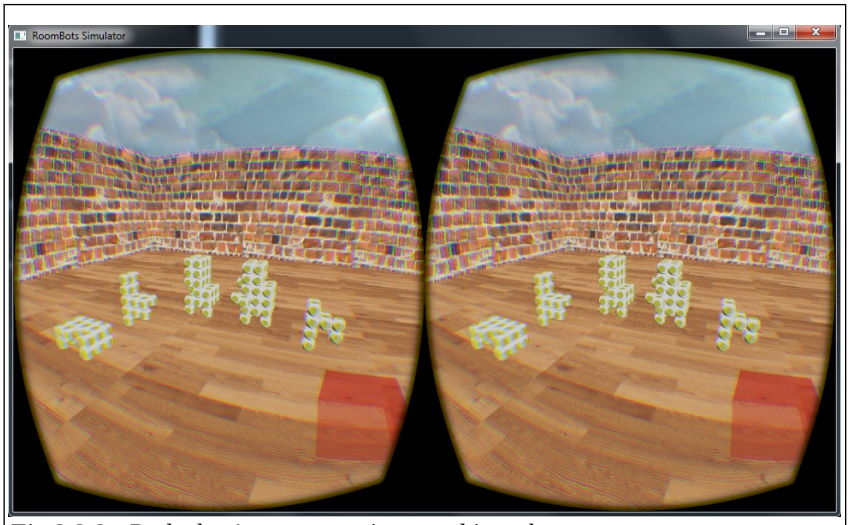

Fig 3.3.2 : Radoslaw's structures imported into the scene

After having my supervisors test out the software as it was, it appeared that it would be interesting to be able to really immerse ourself into the room rather than only standing above it. But the possibility to see all the room at a glance was also interesting, particularly to roughly position Roombots structures and that is when the idea to create two different points-of-view for the scene came up. This led to the implementation of the Box-View and the Room-View. The Box-View makes you feel like you're sitting in front of a table on which there is a box where you can add structures as if they were small objects, while the Room-View feels like you were standing inside the room itself, with the structures in their actual size. In this second view, you cannot add new structures but you can use the WASD keys to move around the room, allowing you to be more precise when moving structures and get an immediate impression of what your room looks like when you're standing in it (See fig. 3.3.3 and fig. 3.3.4 to see the differrence between Box-View and Room-View respectively). It was also decided to bind all structures to a grid where each cell is the size of a half-module so make it much easier to simulate the behaviour of the Roombots. Indeed, the problem of reconfigurating tens of modules in a continuous environment is very complex and binding Roombots to cells of their own size limits the number of configurations and simplifies greatly the algorithms required to perform the desired reconfigurations.

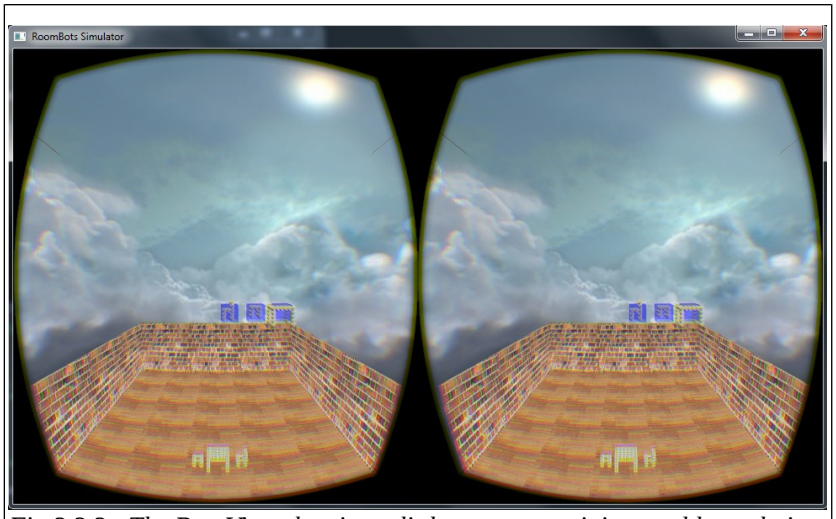

Fig 3.3.3 : The Box-View showing a little scene containing a table, a chair and a stool

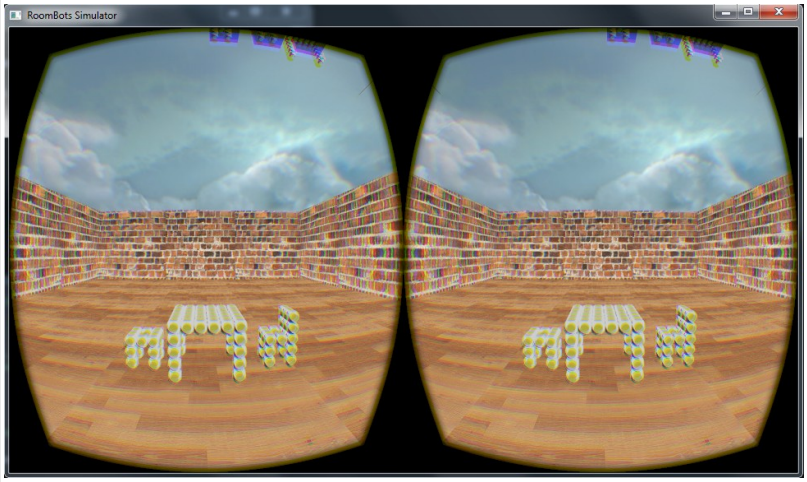

Fig 3.3.4 : The Room-View showing the same scene as in fig. 3.3.3

3.4 Complete immersive interaction framework

Binding everything together, we end up with a complete immersive interaction framework and the first main goal of the project is acheived. Indeed, the software immerses you completely into the scene where you can add furniture anywhere you want before looking around with the Oculus Rift and move around to look at your new room from any angle you want, all of that while sitting in your chair.

3.5 Simulation

Once a room is set up, we want to see how it would look like if you let Roombots free to assemble into the various structures we have placed in our scene. That is the reason why we want to be able to run simulations of such reconfigurations. To do that, the positions of all the Roombots are acquired from the scene and for each of those modules, we use a path-finding algorithm to create a path from some arbitrary starting point to its final position. The path-finding algorithm can be changed easily, as it is simply an object with a « run » method that takes an initial position and a final position and computes the path from one to the other. For this project, and for demonstration purpose, we used a very simple algorithm that doesn't take collision, obstacles or even physics into account but more complex and well-known algorithms such as the Million Module March algorithm or the A* algorithm[27] (a common path-finding algorithm used in a lot of games), can be implemented without changing anything else in the software, as long as they produce a single path for every module. The reason that the MMM algorithm couldn't be implemented was the lack of time left after finishing the interaction framework. Indeed, said framework took most of the semester to finish because it was important that it was the most natural and intuitive possible. As for the movement of the modules themselves, it is really a simple step-by-step vision of it, where one of the half-module acts as pivot and the other one goes from one position contiguous to the pivot to another, thus never « breaking » a module. Conclusively, the simulation looks very nice and it gives a good idea of how Roombots work together to build furnitures and create a believable room.

4. Software testing : User-study

Before the software contained simulation, a software-testing session was held in the lab and ten people tested the software. This sample of people consisted of six members of BioRob, including one woman and all in their late twenties or early thirties, one eighteen-year-old man, one fifty-eight-year-old man and two students, including a twenty-three-year-old woman. Among those people, two of them are wearing glasses. Their general opinion was very positive and the answers to the questions (see appendix A for the complete test protocol) were compiled into the table 4.1.

Question	Average answer
Distance perception	4.1
Improvement feeling	4.3
Intuitive interface	3.9
Immersion	4.5

Table 4.1 : Average answers to all questions. All possible answers going from 1 to 5

Overall, people were very satisfied with the immersion they felt while using the Oculus Rift and the distance perception it gave them. They were also all agreeing they got better at using the software (grabbing and dropping) after using it for three to five minutes and most of them felt like handling structures was intuitive. A few of them had trouble getting the hang of grabbing structures (their hand were often closing but not quite *pinching*). One funny yet interesting observation is that a majority of people tried to stack furnitures. For some reason, they were all expecting to be able to build towers of chairs and stools. It hints at the fact that people tend to expect natural characteristics and behaviours from the virtual environment when using VR. Indeed, when one feels so immersed that it almost feels real, it is easy to expect walls to block you and objects to fall when dropped.

5. Conclusions and future work

Overall, the main goals of the project have been reached and even though we couldn't go as far as the original plan on the simulation side, it is fair to say the development of the interaction framework is a success. It is not perfect, but it is a sufficient proof of concept that hints at the possibilities offered by VR and gesture-recognition when used in combination with modular robotics. Indeed, allowing someone to entirely design a room while sitting in a chair, but still being able to see what it will look like when done sounds like a great improvement in quality of life, especially if you can't move your furniture by yourself. On this point, if further work must be done to improve the life of people with disabilities, it would require some more cooperation with said people in order to ensure the interface is well-suited to their needs. It would probably even be necessary to adapt the software to the various disabilities people can suffer from. For instance, someone with a missing arm would have more difficulty using the software as is, since it is designed to be used with both hands. One other improvement would be to forward instructions to the Roombots so they actually move in the physical world. This would represent the ultimate experience of this project. Set up your room, and then watch the Roombots build it for you. In conclusion, this project uses the latest commercially-available technologies in the human-robot interaction domain to open many new ways to interact with robots.

6. Apendices

Appendix A : Test protocol

This protocol was followed with every test subject and the software was presented exactly the same way to all of them.

Step 1 : Explain controls (keys, gesture)
This software allows you to grab structures with your hand throuh gesture recognition and get a visual immersive feed-back with the Oculus Rift.
You have 2 views : the box view where you sit in front of a box containing the scene, and the room view, where you stand in the scene.
In box view, you can reach for structures and grab them to place them in the room easily.
In room view, you can move around with WASD and also grab structures.
You can switch between views by pressing the space key
The blue cubes represent holders for the structures while the red cube represents your hand.

Step 2 : Try to switch between views and move around in the room
Now, try to move your hand, try to switch views with the space key and try to move around.

Step 3 : Standard procedure with Oculus Rift
Now, you're going to put on the Oculus Rift and do a little exercise.
First, in box view, try to grab structure and drop it somewhere in the room.
Then, switch view and move to get closer to the structure you just dropped.
When you're close enough and you can easily reach for the structure, try to grab it and move somewhere else while still holding it.

Step 4 : Scene set-up
Finally, you can set-up a little scene, for example a table with 3 stools and a chair.

Step 6 : fill short form with a few comments
FORM :

From 1 to 5, how do you think the Oculus Rift helped you perceive distance and grab structures ?

From 1 to 5, to what extend did you feel more comfortable grabbing structures after a few tries ?

From 1 to 5, did you think handling structures was natural ?

From 1 to 5, how immersed did you feel while using the Oculus Rift ?

What is your general opinion on this ? Do you have any comments ?

1 http://www.sciencedirect.com/science/article/pii/S0921889013001632
2 http://biorob.epfl.ch/cms/page-36376.html
3 http://biorob2.epfl.ch/utils/movieplayer.php?id=229
4 http://biorob2.epfl.ch/utils/movieplayer.php?id=275
5 http://biorob2.epfl.ch/utils/movieplayer.php?id=272
6 http://biorob2.epfl.ch/utils/movieplayer.php?id=273
7 http://biorob.epfl.ch/page-53602.html
8 http://biorob.epfl.ch/page-75451.html
9 http://biorob.epfl.ch/page-110120-en.html
10 http://biorob.epfl.ch/page-104432.html
11 http://biorob.epfl.ch/page-122260-en.html
12 http://ijr.sagepub.com/content/27/3-4/331.abstract
13 http://link.springer.com/chapter/10.1007/978-3-642-32723-0_34
14 https://www.opengl.org/
15 https://www.oculus.com/en-us/
16 https://www.leapmotion.com/
17 http://blog.leapmotion.com/hardware-to-software-how-does-the-leap-motion-controller-work/
18 https://git.epfl.ch/repo/roombotssimulator.git
19 http://freeglut.sourceforge.net/
20 http://glew.sourceforge.net/
21 http://glm.g-truc.net/0.9.7/index.html
22 http://www.ogre3d.org/
23 https://developer.oculus.com/documentation/
24 http://www.apache.org/licenses/LICENSE-2.0
25 https://developer.leapmotion.com/documentation/csharp/devguide/Leap_Coordinate_Mapping.html
26 http://www.martinreddy.net/gfx/3d/OBJ.spec
27 http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html