Mobile control interface for modular robots.

Semester project, fall 2010

Gilles Cressier

January 7, 2011

Biorobotics Laboratory

Advisors:
Prof. Auke Jan Ijspeert
Rico Möckel
Stéphane Bonardi
Acknowledgements

I would like to express my gratitude to Prof. Auke Ijspeert for giving me the chance to work on a really interesting topics. I would also thanks Stéphane Bonardi and Rico Möckel for their guidance that keep me on the right track. I know they have spent a lot of time for my project and I am grateful about that. I also express my thanks to all people from the Biorobotics Laboratory and other students doing their project there for the good working atmosphere.
Abstract

Modular robots are robots composed of different modules that can be connected together to respond to the requirements of the moment. The concept of modular robots appears relatively early (1970s), but it was only applied to a portion of the robot. Design a robot that is completely modular is an idea that appears at the end of 1980s, with CEBOT[2]. Modular robotics is an interesting field that is currently in research phase. Maybe in the future they will have various usages, from reconfigurable furniture to locomotion structures. One key point of the democratisation of modular robots is the way they would be controlled. The purpose of this project is to design an interface to control modular robots, especially for the Roombots, and start to implement it.
Contents

1 Project Introduction .............................................. 6
   1.1 Control modular robots .................................. 6
   1.2 Introduction to Roombots ................................. 7
   1.3 Motivations .................................................. 8
   1.4 Methods ..................................................... 9

I Mobile interface for Roombots ................................. 10

2 Introduction ..................................................... 10
   2.1 Target devices ............................................. 10
   2.2 Critical points in designing an interface for modular robots ... 11
   2.3 Related works ............................................. 12
      2.3.1 Eve, the Yamor Simulator ............................ 13
      2.3.2 Molecubes simulation interface ..................... 14
      2.3.3 Player Project ......................................... 15

3 Analyse of inputs management ................................ 16
   3.1 Type of inputs ............................................. 16
      3.1.1 Touchscreen inputs .................................. 16
      3.1.2 Accelerometer input .................................. 17
   3.2 Usage of inputs in applications .......................... 18
      3.2.1 Android (2.1) main page .............................. 18
      3.2.2 Youtube on Android .................................... 19
      3.2.3 Asphalt5 on Ipad ...................................... 21
      3.2.4 Perceptive Pixel: Multi-touch Collaboration Wall 22
      3.2.5 Avogadro, a Molecule editor for Desktop ............ 23
      3.2.6 Instapaper on I-Phone ............................... 24
   3.3 Synthesis: What can we do with such inputs? .......... 25
      3.3.1 One-finger click (\(\ast\)) ............................ 25
      3.3.2 One-finger long pressed (\(\Omega\)) ..................... 25
      3.3.3 One-finger movement (\(\rightarrow\)) ...................... 25
      3.3.4 Two-fingers ............................................ 26
      3.3.5 Multi-fingers ......................................... 26
      3.3.6 Accelerometer ......................................... 27

4 Design interface for Roombots ............................... 28
   4.1 Roombots specificities ..................................... 28
      4.1.1 Requirements of the Interface ....................... 28
      4.1.2 Three interfaces are better than one ............... 29
   4.2 Graphical representation of Roombots .................... 30
      4.2.1 Graphical representation of Roombots: constraints ... 30
      4.2.2 Possible representation of Roombots ................. 31
   4.3 Interactions between the user and the interface ........ 32
      4.3.1 Features requirement .................................. 32
      4.3.2 Fixed interactions ..................................... 33
A User Guide

A.1 List of components of the visualisation . . . . . . . . . . . . . . . 58
A.2 Description of components . . . . . . . . . . . . . . . . . . . . . . 59
A.3 Interactions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 60
   A.3.1 Keyboard interactions . . . . . . . . . . . . . . . . . . . . 60
   A.3.2 Mouse interactions in the visualisation area . . . . . . . . 60
   A.3.3 Mouse interactions out of the visualisation area . . . . . . 60

B Coding convention of the implementation of the QWidget for Roombots

B.1 Naming convention . . . . . . . . . . . . . . . . . . . . . . . . . . 61
   B.1.1 Definition of patterns . . . . . . . . . . . . . . . . . . . . 61
   B.1.2 List of conventions . . . . . . . . . . . . . . . . . . . . . . 61
B.2 Comment convention . . . . . . . . . . . . . . . . . . . . . . . . . . 62
   B.2.1 Type of comment . . . . . . . . . . . . . . . . . . . . . . . 62
   B.2.2 Usage of comment . . . . . . . . . . . . . . . . . . . . . . . 62

C Schedule of the project

63
1 Project Introduction

Modular robots technology is currently in phase of development. Many industries are interested by this technology, and some universities are developing their own robot. People from the Biorobotics Laboratory of the EPFL are currently working on such robots: the Roombots.

![Atron robot](source: [24])

![Roombots](source: [37])

Figure 1: Modular robots

1.1 Control modular robots

Before going further, I would like to precise the notion of control interface. Modular robots - and robots in general - are composed of three types of components:

- Motors
- Sensors
- Other components

By "control", I mean be able to command the motors, and be informed of informations captured by the sensors.

So control in the scope of modular robotics means be able to control either one module, or control the whole robots. An interface to control modular robot is then a graphical user interface that:

1. Allows the user to understand the configuration of the modular robot.
2. Provide some ways of interactions in order to modify the configuration of the modular robot.
1.2 Introduction to Roombots

Roombots project intents to create modular robots that can evolve by themself to construct different furniture structures, for example a chair or a table structure, depending on the users’ needs. This process is called self-reconfiguration. Roombots will also be able to move by themselves, this is called locomotion.

![A Roombots module. Source: [1].](image)

A Roombots Module is composed of two spheres that are paste together[6]. The two spheres can rotate one relatively to the other. Each sphere is composed of two similar parts, obtained by cutting it in two by a symmetry plane, such that 3 faces are on each part. Each face is a connector that can be either active - provided with a latching mechanism, or passive - just a connecting plate[5]. A module is composed of three degrees of freedom:

- One central degree, such that the two spheres can turn one relatively to the other.

- One degree for each sphere, such that the two part of the sphere can turn one relatively to the other.

Each degree is provided with a motor to change the angle of the degree of freedom.
1.3 Motivations

As Roombots prototypes - and modular robots prototypes in general evolve, the necessity of having an interface to efficiently control them increases. By now, software tools for modular robots exist but are generally adapted for only part of the work that can be done with them. For example people from the Biorobotics Laboratory use a robotic simulation environment (Webots[21]), that seem to be well adapted for simulation, and development tasks, but it is maybe not the better tool to control a real modular robot.

So it would be interesting to have a tool that is designed for modular robots, with ability to run either in simulation mode or in command mode, with a two ways communication channel(command the robot and get feedback).

Such a tool should be used during the development process to compare the observed to the expected behavior of Roombots, and provide a way to stop everything in case of failure. Another use of a dedicated interface that can be imagined is a mode in which a lay user can play with the robot, without having to understand how it works.
1.4 Methods

I split the project into two main parts.

**In part I,** I try to determine what could be an ideal interface for controlling Roombots. After having looked at existing works in the same field (subsection 2.3), I make a study over what can be done using accelerometer and touchscreen input. First I study the way of interactions allowed by such input (section 3), and then, I establish the requirements of an interface to control Roombots (section 4). All this work is used to define the specifications of a prototype of ideal mobile control interface for Roombots (section 5).

**In part II,** the goal is to start the implementation of an interface as described above. In this part of the project, I implement a QtWidget (subsection 7.3) that offers a visualisation of Roombots, using Ogre3D (subsection 7.1). The specifications of this implementation are determined (section 8) and the user guide for the implemented widget can be found in the appendix A. Finally, I suggest some possible improvements to the implementation (section 10).
Part I
Mobile interface for Roombots

2 Introduction

In this part of the project, I try to determine what would be a user-friendly interface to control Roombots. Before entering in the core of the subject, I need to take some reflection time, to be sure that I spend my time on relevant topics. Firstly, I try to define what would be hotspots in designing an interface to control Roombots, what are the points on which it would be necessary to think about (subsection 2.2). Secondly, I analyse some interface that have already been implemented (subsection 2.3) to get an idea of what the interface could look like, and how it would be possible to interact with it. Then, I analyse the inputs that can be used by an interface (section 3). After that, I try to precise the design of an interface for Roombots (section 4), in order to define the specifications of a prototype of such an interface (section 5).

2.1 Target devices

The interface to control Roombots - as any other software - will run on a device. As soon as the goal is to have a mobile control interface, the device on which will run the interface has to be mobile. Mobile devices that are currently available are mainly smartphones and TabletPCs. They offers new kinds of inputs, such that accelerometer and touchscreen inputs, but graphical performances and processor resources are significantly lower than those of a modern computer.

![Target devices](image)

Figure 3: Target devices: Smartphones support more inputs than PC, such that touchscreen inputs, but PC are more powerful in terms of computation and graphical performances.
2.2 Critical points in designing an interface for modular robots

Modular robots are quite different from standard robots. They offer many more possibilities, for example two small entities can be merged in one large, or if a piece of the entity is off or damaged, it can be replaced by another, but they have also some drawbacks. Let see what kind of hotspots can be meet during the design process of an interface for modular robots.

![Figure 4: Example of interface to control robotic arm. The interface use a 2D model to represent a 2D robot. Source: [35].](image)

**First difficulty** that can be met is the one of the representation. As modular robots are 3-dimensional object, they can not be just projected on a plan, to be displayed on a screen. What people generally do, is constructing a 2-dimensional model that represent the 3-dimensional state of the object. This approach works well in the case of non-modular robots, where the model can take into account the connection between subcomponents, and possible states of each of them. Modular robots are a little bit harder to reduce in a model, because they can be reconfigured into a large number of structure, and the model has to guarantee that it can represent each of them. Such model for modular robots are feasible, but in fact, they are not well readable for a non-expert user.

**Second difficulty** with modular robot is to separate general movement (of the whole robot) into sub-movements (understandable by each servos). This problem is known as inverse kinematics[31]. Inverse kinematics also appears in the field of robotic arm. For example, a robotic arm try to take an object in its hand. The first step consists of moving the hand on the object. The position of the hand is easy to determine given the angles of the shoulder and the elbow, but determining the angles that shoulder and elbow has to take, in order to move the hand at a given position is more difficult.

Inverse kinematics is not in the scope of the project, but it’s important to see that an interface for modular robots can not just assign specified angles to joints, but have also to decompose a "hand position" to "joints angles", in such a way that the user can use the robot as he uses his own arm.
2.3 Related works

In this subsection, I analyse different software that are related either to the visualisation of modular robots or to the ways of interactions with them. I select three softwares. The first one (Eve, subsection 2.3.1), is a simulator for YaMoR. It is based on script. A configuration of YaMoR and a set of movements of its components can be written in a script, and the simulator will read the script to generate the simulation. What I found interesting is that developers used a video game map as simulation environment.

The second software I analyse is a simulator for Molecubes[7](subsection 2.3.2). Molecubes are relatively similar to Roombots, so I think that analyse this interface can be valuable for further parts of my project.

As the two firsts software analysed are purely simulator, and for a specific modular robot, I choose a more versatile software as third, the player project (subsection 2.3.3).
2.3.1 Eve, the Yamor Simulator

YaMoR\cite{39} is a modular robot developed at the BIRG\cite{26}. Eve is a simulation software implemented in the scope of a semester project by Cyril Jaquier and Kévin Drapel, in 2005, based on previous works of Yvan Bourquin and Daniel Marbach, and on Bluemove\cite{10}. Eve is written in C++, using ODE\cite{15} as physic engine and Irrlicht\cite{13} as render engine.

The concept is that they put a YaMoR robot in a Quake III\cite{20} map for the simulation.

![Figure 5: Eve: a simulator for YaMoR. Source: \cite{38}](image)

**Strengths of Eve:**

- The simulator use a physic engine to be more realistic.
- Putting the robot into an elaborate environment (Quake III maps) is interesting and valuable, since a lot of map editors already exist to generate such environment.

**Weaknesses of Eve:**

- The only way to interact directly with the robot representation is to move it in the map before the simulation, using the keyboard. Every other interaction has to be filled in a script that the application execute during the simulation.
2.3.2 Molecubes simulation interface

Molecubes[7] is an open-source modular robots project. A Molecubes simulation interface - called Molecubes software[14], has been implemented by Andrew Chan under the supervision of Viktor Zykov of the Computational Synthesis Laboratory, Cornell University. The Molecubes software use AGEIA PhysX[17] as physic engine and Ogre3D (section 7.1) as render engine.

![Figure 6: Molecubes software. Source: [14]](image)

**Strengths of Molecubes software:**

- The simulator use a physic engine to be more realistic.
- Visualisation is realistic, that is an advantage to elaborate realistic videos, and it might also be valuable for the end user to have a good visualisation.

**Weaknesses of Molecubes software:**

- The ways of interactions with the visualisation is not intuitive. (What mean ”rotate cw”? Where will ”Add cube pos y” effectively add a cube?..).
- Selected object are marked by displaying their bounding box (the white cube on the screenshot). I think this is not the best way to do it, because it is not intuitive. A small amount of time is needed to determine which object is selected. My opinion is that changing the color of the object would ease the action of the user to found the selected object.
2.3.3 Player Project

The Player project[18] is a tool that can be used with any kind of robots. It is divided in three modules:

- **Player** is used to communicate with robots. Based on client/server model. Support multiple programming languages to control the robots.

- **Stage** is a simulator for multiple robots in a 2-dimensional environment.

- **Gazebo** is a simulator for multiple robots in a 3-dimensional environment.

The three modules are compatible, so for example Player and Gazebo can be used together to control the robots in real-time.

![Figure 7: Gazebo. Source: [27]](image_url)

**Strengths of Player Project:**

- A lot of feature have been implemented, it’s really a large and complete solution.

- Is widely used, so it is documented, approved and constantly evolves.

- Separation of simulation and communication is good if the target platform is restricted on resources. So if you just want to communicate with the robots, you don’t have to do all the computation for the simulation.

**Weaknesses of Player Project:**

- Adapt the software to requirements of a modular robot can be tedious, and the result can be really not optimal in term of resources consumption.

- This is typically a development interface, ideal for simulation and test, but not really adapted for inexperienced users.
3 Analyse of inputs management

In this section, I study the inputs that can be used to interact in a software on devices of new generation, such that a mobile phone or a tabletPC. This will be useful to construct the bases of an interface, because I have to take into account the ways of interaction that can be used to control the robot.

3.1 Type of inputs

Inputs I will cover are touchscreen and accelerometer. They offer a lot of possible interaction, so I will use a methodology to analyse them. I will not cover the mouse and the keyboard inputs, because the ideal interface would be designed to make usage of interactive device such as TabletPC or smartphone, but touchscreen and accelerometer inputs can be emulated with a keyboard and a mouse if necessary.

3.1.1 Touchscreen inputs

Convention I will define convention to represent what do fingers do on the screen.

- Indicate that a finger click.
- Indicate that a finger stay pressed at a fixed position.
- Indicate that a finger stay pressed and move in the direction of the arrow.

List of touchscreen inputs The first type of input on touchscreen are one-finger inputs. With one finger, possible inputs are click( ), stay pressed ( ) or describe a movement ( ). The second type of inputs allowed by touchscreen are two-fingers inputs. The two fingers can move or stay pressed in a fixed position. For example, the user can move his finger parallel ( ), antiparallel ( ), one move the other not ( ), and so one. Finally, there exists multi-finger inputs. The idea is the same as on the two-fingers, but less screen provide such inputs.
3.1.2 Accelerometer input

An accelerometer is a sensor in the device that provide information about the positioning of the screen. General accelerometer can say what acceleration is applied to the device. This can be used to determine the orientation of the device, under the assumption that the device is not moving (using gravity).

Convenction I’ll also define a convention to represent accelerometer input. First, I’ll consider two possible initial positions:

**H** as Horizontal. Imply that the screen position is considered relatively to the horizontal position. Screen at the horizontal is equivalent to "no input".

**S** as Start position. Imply that the screen position is considered relatively to the position at which the application started. Screen fixed is equivalent to "no input".

Second, I’ll define the variation relatively to the considered position.

→ Indicate that the slope goes down to the right of the screen.

← Indicate that the slope goes down to the left of the screen.

↑ Indicate that the slope goes down to the top of the screen.

↓ Indicate that the slope goes down to the bottom of the screen.

For example, if my screen is in H→ position, then water on the screen will flow to the right.
3.2 Usage of inputs in applications

In this subsection, I examine different software applications that make use of accelerometer and touchscreen inputs. I choose them in order to have a range as wide as possible. My panel is composed of applications that are widely used and user-friendly, as Android main page (subsection 3.2.1), and Youtube on smartphone (subsection 3.2.2), of application that are more dedicated to a specific segment of the population, as Avogadro (subsection 3.2.5). I select also applications that manage a 3-dimensional visualisation, such as Avogadro, and Asphalt5 (subsection 3.2.3). I also try to cover all the possible inputs, that is why I have decided to investigate Instapaper (subsection 3.2.6), that make a interesting usage of the accelerometer, and the Collaborative Wall (subsection 3.2.4) that try to explore new fields of interactions with touchscreen using multi-fingers inputs.

3.2.1 Android (2.1) main page

Android[8] is a smartphone operating system provided by Google[28].

The main page of Android 2.1 is composed of seven views. The default view is the fourth. There are two ways to switch from one view to the other. First ← (→) can be used to go to the next (or previous). Second zoom out (↘↖) can be performed to obtain an overview of the seven views, then the wished view is obtained by clicking on the corresponding item in the overview.

![Android main view](image)

(a) View 4 (default view)  
(b) Overview of the 7 views

Figure 8: Android main view.

In each view, they are some items (Widgets). The application associated to the widget can be run by clicking on it (○), and the widget can be selected - to move or remove it - by long pressing on it (汜).

18
There are also four special fields on the screen, a notification board and 3 buttons. The notification board is at the top of the screen and contains small pictograms. To get informations, we have to lower it as a roller blind (↓ from top of the screen). The three buttons are at the bottom of the screen. The first provides access to a list of all applications, the second opens the phone application, and the third is used to add a widget to the current view. This last feature is also obtain by long pressing (Ξ) where there is no widget.

The accelerometer isn't used in this main page.

So if I count how many different touchscreen inputs are active on this page, I obtain 5 one-finger (®, →, ←, ↓) and 1 two-finger (↖, ↘, ↘), where 3 are position-free (→, ←, ↘) and 3 are position-dependent (®, →, ↓). I think that 5 active touchscreen inputs on the same page are quit a lot. In fact, Android-phone is designed for day after day usage, so it has to be as ergonomic as possible. The complexity is partially acceptable in case that the whole system is homogeneous - at best one semantic per inputs, more realistically similar actions for a same input.

**Strengths:** Clearly, the main advantage of this system is that - when you know how it works-, it take approximately no time for doing what you have to do. For example, on my configuration of Android on my phone, how many inputs do I need to reach some goal?

<table>
<thead>
<tr>
<th>Action</th>
<th>Number of inputs from the main page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get the time</td>
<td>0</td>
</tr>
<tr>
<td>Search on Google</td>
<td>1</td>
</tr>
<tr>
<td>Call my girlfriend</td>
<td>2</td>
</tr>
<tr>
<td>Read my e-mails</td>
<td>2</td>
</tr>
<tr>
<td>Get weather forecasts</td>
<td>2</td>
</tr>
<tr>
<td>Switch off the wifi</td>
<td>2</td>
</tr>
<tr>
<td>Take a picture</td>
<td>1</td>
</tr>
<tr>
<td>Get my scheduler</td>
<td>2</td>
</tr>
</tbody>
</table>

**Weaknesses:** One of the main weakness of this system is that it need some experience to be well used. There are no way - except web documentation - to learn the actions associated with the different inputs.

### 3.2.2 Youtube on Android

Youtube[40] is a website to broadcast videos, and play broadcasted videos. An application for Android has been developed, such that videos from Youtube can be played on an Android smartphone.

The main page of youtube on Android show us a list of video. The page fill the screen widely and there is a scroll to go down in the page. We can roll up or down the page by ↑, ↓. To play a video, the user just has to click on its description (®). Another option the user has is to open a menu (comments/bookmark this video,...) by long pressing on it (Ξ). There are no two-fingers inputs that are implemented on this page.
The user can use the accelerometer to switch between portrait/landscape view of the page. H← set the view to landscape. Any other configuration set the view to portrait.

![Figure 9: Youtube on Android.](image)

I see that this page contains 4 one-finger touchscreen inputs (,column, ↓, ↑). Two of them are position free (↓, ↑) and two are position dependents (, ). I also see that an accelerometer input is implemented.

I think this application is coherent with respect to the system in which it take place (i.e. Android, subsection 3.2.1), because most of the inputs have a similar semantic.

**Strengths** One of the main advantage is the coherency with Android. Someone who have the habit to use his smartphone won’t encountered difficulties to use the Youtube application. Another point is that there are few inputs that are active, so it’s easy to remember how it works, even if the application is seldom used.

**Weaknesses** I think this application is well done, but one point that I would change if I was in charge of it, is the default view. I would add the option of searching directly for videos from this view. Every time I use this application, it’s to find a particular video, and I don’t care about what video is the most viewed. Suggestions are good, but I think it’s less important that search capabilities.
3.2.3 Asphalt5 on Ipad

Asphalt5[23] is a race game over Apple devices[22]. The interesting part in this game is the page to choose the car, and its settings. The user can see a 3-dimensional view of the car in the background.

In the foreground of this view is displayed a menubar with 5 buttons on top, and 3 buttons on bottom. Buttons on top are related to the option the user has (settings of the car), and buttons on the bottom are more generals (quit the game, go to the race, infos).

What I found interesting in this view is the way the user can move the camera. The camera - point from which the user see the car - is placed on a half-sphere that is centered in the car. With one finger, the user can move the camera around the vertical symmetric axis (→, ←), and the user can also change the latitude of the camera (↓, ↑). The user can combine this two movements in one (↗, ...). When the user stop pressing, the camera continues its movement a short instant to make it more fluid.

Strengths I see that the view of this menu is really simple. There are only one-finger inputs, and those inputs are only click on buttons (o), and long pressing movement (e.g. →) to move the camera, but it looks well I think. The fact that the user can move the camera as he wants give him the feeling of having an application that is complete, fun, and well-done.

Weaknesses I think that having more interactions with the car would be really valuable for this scene. For example, the user would be able to click on a portion of the car to open a menu that allows to modify the settings (if he clicks on the bodywork, he can choose the color, if he clicks on a wheel, he can adjust the wheel-suspension, and so one), instead of using an option menu.
3.2.4 Perceptive Pixel: Multi-touch Collaboration Wall

The Multi-touch Collaboration Wall is a screen/board that allows several persons to work over the same touch-screen. It is developed by the company Perceptive Pixel[33]. In the United States, it is also known as "CNN’s Magic Wall", because CNN use this screen in a political emission. It is difficult to figure out what input are used based on video, but here is a list I tried to do as complete as possible, based on the advertising video.

![Perceptive Pixel: Multitouch Wall](image)

Figure 11: Perceptive Pixel: Multitouch Wall. Source: [33]

→ Move the object that is under the finger in the direction of the arrow.

☐ Get a menu/dialog.

☐ \(\_\_\) on object  Zoom in/Turn. To zoom: The fixed finger has to be in the axis of the arrow, without what the object turns (Fingers stays at the same point of the object).

☐ \_\_ Draw a rectangle of selection (as long-left pressing with a mouse).

☐ ☐↑ Move the point of view (camera).

**Strengths**  The advantage of this solution is that there is an amazing number of ways of interaction.

**Weaknesses**  I don’t know how long it takes to be familiar with this tool, but for sure, you can’t use it before learning a few. Video from Perceptive Pixel show us a lot of features, but in fact, when I look at a real usage - not a demo -, people use just a few of those. For example, the presenters of CNN that use it never make use of advanced features - in fact, they use it almost as a television. Why? Are advanced features buggy? Or didn’t they understood how it works?
I don’t know. But what I see is that the collaboration wall isn’t used in the way that it has been designed for, and that is a sign of failure. This is the reason why I think the collaboration wall is a nice way of exploration, but it is not a ready-to-the-market product.

3.2.5 Avogadro, a Molecule editor for Desktop

Avogadro is a Software to draw molecule and detail some chemical specifications. It runs on computers and the standard input device is the mouse, that is the device I use for study the interactions.

The main part of the graphical user interface of Avogadro shows us a 3-dimensional view of the molecule. The user can make the molecule turn around its center by drawing a vector (keep left press on the mouse and move). The molecule is treated as if it was in a sphere of center its center of gravity. The vector the user gives will be applied to the sphere at the point that is the closest to the user’s eyes. So for example if the user draws $\rightarrow$, the sphere - it is invisible but you can imagine it containing the molecule - will turn as the earth viewed from space. The size of the vector gives the frequency of rotation.

![Avogadro interface](image)

Figure 12: Avogadro: Software to edit and visualize molecules. Source: [25]

**Strengths** The option of making the molecule turning is helpful to understand its structure. The user have a better idea of how its atoms interact one with the others.

**Weaknesses** The way of defining the rotation ($\rightarrow$ with the mouse), is not completely intuitive.
3.2.6 Instapaper on I-Phone

Instapaper is an application for I-Phone that allows the user to save Web pages/texts to read them when he has the time.

When the user opens instapaper, he can display a text he had saved before. The point I found interesting is that the user can scroll up/down (to go through the text) two ways. First, he can use a touchscreen input (↑;↓). Second, he can use the accelerometer. He can tilt the I-Phone back (S↓) to scroll down, forward (S↑) to scroll up. When the user opens a file, the phone saves accelerometer data (as a zero position). After that, it compares the current accelerometer data with the zero position data. The difference of slope gives the scrolling velocity.

**Strengths**  The way of going through the text (using accelerometer) is not usual, and is easy to assimilate, it seems almost natural.

**Weaknesses** One drawback that I see is that the user has to be static to read it. Otherwise, for example if he is in a train that accelerate, he has to counteract to keep the text stable, because the phone can’t make the difference between acceleration of the train and acceleration due to a relative movement of the screen. Another point is that when you read a small paper, it sounds good to use this feature, but when you read a five hundred pages book, your wrist will become painful.
3.3 Synthesis: What can we do with such inputs?

3.3.1 One-finger click (○)

I think that it is commonly defined - because most application use this way (subsections 3.2.1, 3.2.2, 3.2.3) - that one-finger click will be used to click on something clickable (button,...) or select an item on a view. Due to the past experiment of users, associate one-finger click (○) with click/select/deselect action will not seem complex. So I think it will be a bad idea to associate it with something else.

3.3.2 One-finger long pressed (□)

As I have seen in android main page, long pressing can be used as the equivalent of the right-click on a mouse - to provide access to a menu, select the object for moving or removing it. In Youtube, such input was used to call a menu (associated to the video on which appears the input) where the collaborative wall opens a menu/dialog. Asphalt5 doesn’t use such input in its menu.

This action seem intuitively adapted to open a menu. Another action that could be associated to such input is a state changing. For example, suppose that an object drawn in the view can take few different states. I can imagine that when the user long press on it, it changes its state (every second). For our example, lets take object O, that can be in state A, B or C. Suppose it is in state B. So if the user long press on it, after one second it will change it state to C, after 2 second to A and so one. A pictogram can show the current state to the user.

3.3.3 One-finger movement (→)

I think that associating this input to an action is one of the major decision in the design of an interactive interface. It is a crucial point, and it is really important to choose the one that better match the requirements. First lets analyse how studied applications use it.

In Android main page, → and ← are used to switch from one view to the other, where ↑ and ↓ are not used.

At the opposite, Youtube and Instapaper use ↑ and ↓ to go up/down the page.

Asphalt 5 main menu (and Avogadro) implements a change of point of view (respectively a rotation of it) when the user applies such input.

For the collaboration wall, this input implies a movement of an object from the source to the destination of the arrow.
3.3.4 Two-fingers

As a lot of combinations can be done with two fingers, I will sort them by type.

**Two-fingers click (žž)** One of the usage I can imagine for this interaction is a multi-selection. For example, if the user wants to apply an operation over two (or more) elements -that can be reflective (e.g. delete them), symmetric (e.g. swap them), or correlated (e.g. add edge between them), I can click simultaneously on both to open a dialog presenting possible operations.

**Two-fingers statics (¤¤)** I saw that neither Android, Youtube, nor Asphalt make use such input. I think the reason is that it seem not a common action: every meaning of this input will be surprising for the user.

**Two-fingers antiparallel (↖↘, ↘↖)** The major usage of this kind of inputs is to zoom in/out. In fact, I didn’t find any other interpretation of ↖↘, ↘↖.

**Two-fingers parallel (↑↑)** In the Android standard list (that I have not studied in depth, because it is almost the same as Youtube), the ↓↓ and the ↑↑ are implemented to manage absolute scrolling. For example, if you want to reach an element in the list that is approximately at its three-fifths, you can ↓↓ / ↑↑ from anywhere to the three-fifths of the screen to browse the list quickly. Remark: The user can also directly click at the right place (žž).

**Two-fingers mix** If wanted, exotic inputs can be used to get more interactivity with our device. The Collaboration Wall is using them a lot. The ones I found are select-zone (že ↓), and rotate object (¤↓).

I think that it’s not a good idea to base our interface on exotic inputs because it requires that the user have to learn them.

3.3.5 Multi-fingers

Multi-finger inputs can typically be used to multi-selection, as I suggest in 3.3.4, or to change the point of view, as it has been implemented on the collaboration Wall.
3.3.6 Accelerometer

I saw that accelerometer input can be used to set the orientation of the screen (Youtube on Android, subsection 3.2.2), or scroll down a page (Instapaper, subsection 3.2.6).

Can we do another usage of such input? I think it will be nice to use it a little bit deeper. For example, I can think at it as a way to change view point. This will maybe be uncomfortable because each time I move the screen, my view change.

An improvement of such method is to add a button/touch-zone on the screen that allow the user to activate (when\(\uparrow\)) the input. The position at the beginning of the pressure will be the position in which the view doesn’t move. If the user move to another position, this will imply a velocity of the camera. So the user can use the screen as a joystick to set the camera at the point he wants.
4 Design interface for Roombots

4.1 Roombots specificities

4.1.1 Requirements of the Interface

In the previous section, inputs have been analysed to see how they can be used. The next step is now to determine what are the requirements of the interface, what features it has to provide. That is what I do in this section. After that, I will use this two analyses to define the specifications of the interface to control Roombots (section 5).

Firstly, the interface should give a readable view of the roombots. Secondly, the interface has to provide efficient ways to communicate with the robots. Finally, it will offer the possibility to the user of stopping everything in case of danger for the integrity of the Roombots.

Before starting the description of the requirements of the interface, I need to introduce some notions.

- **Couple** A couple is a set of degrees/couples. Each element of the set is associated in the set with a ratio. When the user orders an action to the couple, the ratio will determine the action that will be ordered to the associated degree. For example, suppose degree A and degree B are coupled in couple C, A with the ratio 1, B with ratio -0.5. Suppose now that the user orders to C a rotation forward of velocity 4 during 3 seconds. Then the order will be distributed to A and B such that A will rotate forward with velocity 4 during 3 seconds, and B will rotate backward with velocity 2 during 3 seconds (synchronously).

- **sequence** As its name suggests it, a sequence is a set of time-dependent actions. A sequence also contains a set of assertions (over the initial configuration/position) the roombots has to fulfill.
4.1.2 Three interfaces are better than one

The software has two main goals. The first is to provide a tool for people who work with the roombots to make tests easier (for example, if they want to check if a sequence of movements is feasible or not). The second is to offer an efficient ways of showing demo. For example, when there are open doors, visitors want to play with the roombots by themselves. So the idea is to provide an interface that is easy to use, and appealing.

- **Low-level interface**
  This interface will mainly be used by people from the biorobotics laboratory. An important point is that the user would define specific actions. By specific actions, I mean that every mechanically possible action can be ordered. For example, if he wants to apply a rotation of three-fifth of degree A followed by a rotation of one-half of degree B, he would be able to do it. Another point is that the user can get any information he wants, i.e. he can read the value of degrees of freedom, see the feedbacks from sensors if the interface is connected to the real robot, and draw graphs based on those data.

- **High-level interface**
  This interface will mainly be used for demos. It has to be as intuitive as possible and hide most complexity of the system behind, including inverse kinematics.

- **Sequence editor interface**
  This interface allows the user to create sequence (of actions). Sequence can after that be used in low-level and in high-level interface. Sequences will be preemptives. So for example, a sequence S1 connects two modules, a second sequence S2 moves two connected-modules, and a third S3 disconnects two modules, the user would be able to create a sequence S4 that uses S1, S2 and S3 as subsequence to connect, move and disconnect as a whole.
4.2 Graphical representation of Roombots

4.2.1 Graphical representation of Roombots: constraints

One crucial point is to find an efficient way to represent the Roombots on a small screen. I have to take care of different elements. Here is an enumeration of those.

**Readability** I won’t overload the small screen I have to work with.

**Ease of understanding** A naive user has to understand how is configured the Roombots (characteristics of its structures: how many modules composed it, where are there, what is connected to what) in the same time he will do looking at the real robots.

**Computation resources** The interface will be used on several kind of devices, and some of them don’t have powerful graphic card.

**Accessibility** Every component on the view must be accessible by the user. I can’t say ‘The module the user wants is hidden by another, and there are no way to bypass it, so the users should use the ones that are reachable.’

In all the next representations, I assume that user can select module, connector and degree of freedom to get some dialog over possible action he can do.
4.2.2 Possible representation of Roombots

2D representation I can imagine a 2-dimensional view of the robot. Each module is represented by a circle containing the value of its three degrees of freedom. Arrows will indicate the connections. A convention to know between which connectors modules are connected would be required.

3D perspective The main idea is to represent robot as a 3-dimensional element in an environment (grid or else). I will have a representation of space, in which I will place the roombots and the camera. The degrees of freedom will be represented by a line describing their axis. The user can move the camera and zoom in/out as he wants, using touchscreen or accelerometer inputs.

Axonometric projection Axonometric projection is a draw where each couple of lines that are parallel in the reality are also parallel in the representation. This kind of view are more readable for the user and easier to compute than 3D perspective, but the weakness is that the view can be misinterpreted in certain particular case.

Descriptive geometry Descriptive geometry permit to obtain 2D-view from 3D-object. The user can for example see the object from the front, the side and the top. I can imagine to have a view of front of roombots, so the user sees each sphere that is at the first plane. If the user wants to see those that are in the second plane, he can zoom in. The first idea was to represent elements that are in other levels semi-transparent, but in fact it’s not really a good idea because the interface will lose readability. Another manner to make clear where our level is, is to put - at a corner of the screen - a small representation of the whole robots, with a line representing the current level. The advantage of this representation is that it is easier to draw if object are basically placed. The main weakness is that if an object is between two plane, how will I draw it? Will I introduce an intermediate plane between the first and the second? If the Roombots were always in squared position, it will be a good option, but I think it is not adapted for the requirements of the interface.
4.3 Interactions between the user and the interface

In this section, I first try to determine the interactions the interface has to provide (subsection 4.3.1). After that, I present interaction that are not flexible (subsection 4.3.2) - i.e. that have only one way to be implemented.

4.3.1 Features requirement

As I have seen before (subsection 4.1.1), the (real-time part of the) interface has to provide a way to stop everything in case of danger. I want to allow the user to select one or more module, to get information or to order an action. The user will be able to switch from one interface to the other.

To be as clear as possible, I will try to list all the feature that the three interfaces (subsection 4.1.2) will need. (? indicate features that are not crucial)

<table>
<thead>
<tr>
<th>action</th>
<th>low-level</th>
<th>high-level</th>
<th>seq. editor</th>
</tr>
</thead>
<tbody>
<tr>
<td>stop</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>quit</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>load a representation of RB</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>? add module to the representation</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>(multi)select module</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>(multi)select couple</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>(multi)select degree</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>? CTRL- equivalent for multi-selection</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>? switch to other interface</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>get low-informations</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>? change representation mode</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>? change view point</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>? zoom</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>change preview time</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>order action</td>
<td>yes</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>order sequence</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>add action to sequence</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>add subsequence to sequence</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>save sequence</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>couple degree/couple</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>uncouple couple</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>change level or graphic details</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>change graphics quality</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>
4.3.2 Fixed interactions

Why defining fixed interactions? I think that some interactions have only one way to be associated to an input because of many reasons. For each interaction I fix, I will give those reasons as justification.

**stop** Since this action is an emergency interaction, it’s important that the user know it, and is able to order it quickly. And I think putting a stop button on the interface is clearly the best way to guarantee that it can be quickly performed and every user know its existence.

**quit** As stop, quit will also be represented by a button. The reason is that if someone that never used a touchscreen before and open this application unfortunately, the interface will provide him a way to quit it.
5 Specifications of a prototype of interface for Roombots

Until now, I have looked at already implemented interfaces to control modular robots (subsection 2.3), then I made a study of the usage of accelerometer and touchscreen inputs in existing application (section 3) to see how interactions can be implemented. After that, I analyse the requirements of a control interface for Roombots (section 4). In this section, I try to define the specifications of a prototype of interface for Roombots, based on the previous works.

5.1 General concepts

5.1.1 Structure of the view

The idea of this prototype of interface is to be composed of four modes. Each mode defines the interactions between the user and the interface and also impact on what is displayed.

The view will be composed of two kinds of elements. First, in the background, is placed the visualization of the Roombots, in its environment. Second, at a higher layer, a set of elements characterizes the mode in which the user is.

5.1.2 Modes

The interface will present four modes:

- **Assembly mode** Used to add module to the visualization, set the environment (grid, plan, ...).
- **low-level mode** Also known as the low-level interface, or expert mode.
- **high-level mode** Also known as the high-level interface, or standard mode.
- **Sequence editor mode** Used to construct and manage sequence.

5.1.3 Notion of context

The context is a set of parameters that will define the aspect of the view, the interactions allowed and their interpretation.

The context is made of the following elements.

- The mode is the most important element of the context.
- The environment (locomotion, or reconfiguration on a grid).
- Is the interface connected to the real robot or not.
- The detail level (How much details will be represented).
- The graphic level (What quality of visualisation does the user want.)
5.2 Visualisation

For the representation, a 3D perspective (subsection 4.2.2) will be used, because it is the only representation - between those studied - that guarantees the readability in the case of Roombots.

Two ways of interactions are allowed with the visualisation. First the user can zoom in (↖↘)/out (↘↖). Zoom in depends on the position of the fingers on the screen when zoom is ordered. For example, if the user zoom in using ↖↘ on the bottom-left part of the screen, this will result with a zoom in the bottom-left part of the visualisation.

Second, the user can turn the camera (→, ←, ↑, ↓) around a point that is relatively fixed to it (basically in front of the camera).

The visualization will contain three basic components:

- **Overview tab** The overview will be reachable as the notification board on Android (subsubsection 3.2.1), but at the left side of the screen. To show the overview, the user has to ← on the tab, where → on it will hide the overview. An optional feature is that when the user zooms in, the overview will automatically be showed.

- **Context tab** A tab to indicate to the user in which context he is - that can be used to switch mode (subsection 5.8), view context setting and change them.

- **Quit button** A quit button will also be present on the screen to leave the application.

![Visualization with overview tab closed.](image1)

(a) Visualization with overview tab closed.

![Visualization with overview tab opened.](image2)

(b) Visualization with overview tab opened.

Figure 13: Visualization.
5.3 Menu bar

I will add a menu bar to the interface, depending on the context. The menu bar will be contained in a tab at the right of the screen. This tab - exactly as the overview tab - will have two states, hide and showed.

To get the menu, the user can open it (using $\rightarrow$ on it).

When the user clicks on an object on the screen, the menu bar will automatically be showed to display the corresponding menu (depending on the context and the selected object).

To hide the menu bar, the user has to $\leftarrow$ on it.

The menu will allow to do everything that can be done in the corresponding context, whether some features will be provided by other ways.

(a) Menu bar closed and stop button (low/high-level context)  
(b) Menu bar opened and chronograph (Sequence editor context)

Figure 14: Menu bar

5.4 Assembly mode

The assembly mode will add an assembly menu, contained in the menu bar. This menu will allow the user to:

- Add, remove modules (using drag and drop) on the current representation.
- Set the environment (grid, other, nothing).
- Open a representation from a XML-file.
- Save current representation to a XML-file.
- Create new representation.
- Associate the representation with the real Roombots (defining modules-ID, ...).
5.5 Low-level mode

Graphically, the low-level mode will add a low-level menu in the menu bar (subsection 5.3) and a stop button to the view. The stop button will be placed in the bottom-right corner of the screen. It will allow the user to stop everything on the roombots. The low-level mode will offer a lot of low-level functionalities to the user:

- Connect single ACMs.
- Get/Set the value of a degree of freedom.
- Read sensors and plot graph with related data.
- Run sequence.
5.6 **High-level mode**

The high-level mode will add a high-level menu (subsection 5.3) and a stop button to the view.

The stop button will be the same as in the low-level mode.

The high-level mode offer less functionalities as the low-level mode, but the concept is that it offer it in a more visual/interactive way.

To introduce this, I will present two examples of such features.

5.6.1 **Locomotion suggestion**

Suppose that you have a locomotion configuration, the only thing you want to do is to move the whole robot in a given direction. In other word, you don’t care about the set of instructions that will lead you from A to B, you just want to go to B.

So to make easier the work of the user, the interface will graphically suggest him some sequence that can be applied to the situation.

Figure 15: The visualisation is augmented of visual representation of sequence(arrows in this case) that offer the user an easier way to order movement to the roombots. Source of the original image: [36]

5.6.2 **Reconfiguration suggestion**

Suppose that you have a module on the grid, at a certain position, and you want it in another position.

One efficient way to order the movement will be to select the module and give it its destination (using drag and drop for example).
5.7 Sequence editor mode

As every mode, sequence editor will add a specific menu to the view (in the menu bar, section 5.3).
It will also add a chronograph (a time manager), at the bottom of the screen. This chronograph allows the user to have an overview of actions that take place in the sequence, and visualize them in time.

Actions in chronograph will be represented by rectangles. The user can select an action by clicking on it.

![Figure 16: Chronograph.](image)

The chronograph also display a line to show the current time visualised. The user can change this "preview" time by long pressing on another point in the chronograph. Another way to change preview is to define it in the context menu. This solution is more precise because it’s easier to select a value than long-pressing to the right place.
5.8 Switch between mode

As soon as I have to display different modes on a single screen, I have to define a policy to switch between them. The way I will manage this is directly inspired by Android main page (subsection 3.2.1). The modes are ordered and the user can switch to the previous (the one that is directly at left of the current) or to the next. To do this, the user will use two-fingers movement.

→→ To switch to the previous (left) mode.

←← To switch to the next (right) mode.

Figure 17: Changing context.

The order will be as followed:

1. Assembly (mode 1)
2. Sequence Editor (mode 2)
3. Low-level (mode 3)
4. High-level (mode 4)

Another way to switch mode will be to change it in the context tab. A third one will be to click (>Lorem) on little arrow at the side of the screen.
5.9 Example of usage of the interface

I have seen that the interface can suggest sequence to the user (subsection 5.6.1). Let’s see a little bit deeper how such suggestions can be added to our interface. First the user has to construct a configuration in the assembly mode to represent the roombots in the initial position he wants.

To do this, he has to define the environment (a grid or a floor). After that he can add modules to the representation, one by one, and specify what is connected to what. This configuration can be saved in a xml-file.

Then the user can switch to the low-level mode to do some tests. For example, he can define how much torque he need to make each movement of the sequence he want to build.

When he is fixed on how the sequence will be composed, he can switch to the sequence editor mode, to properly edit the sequence, adding actions in the chronograph, and simulate it.

Now he has a first version of the sequence. He can save it and switch back to the low-level mode to try it on the real robot. If changes have to be done to the sequence, he can go back to the sequence editor and perform them. he can iterate this process until the sequence is exactly what he expects.

Then, in the sequence editor, he can associate a graphical object to the sequence (in our example an arrow).

Finally, in the high-level mode, if the configuration is similar as the initial position of the sequence, the interface will suggest the sequence by displaying the graphical object associated to it. By clicking on it, he can run it.

Figure 18: Process of creating a sequence.

---

1. finding similarity is not in the scope of this project.
6 Conclusion

As I have seen (subsection 2.3), tools to control modular robots exist and are mainly focused on development. My opinion is that developing tools that are more focused on usage would be a necessity in the evolution of modular robots technology. By now this part is undervalued but for the end-user, the way to control the robots would be as important as the features that the robots can have.

I think that constructing a user-friendly interface to control the Roombots is feasible, and it would be a valuable component of the whole Roombots Project, in comparison to other modular robots project.

One of the crucial points of such a user-friendly interface would be how the user will interact with. New generations of devices - such as smartphone or tabletPC - allows interesting inputs, such accelerometer and touchscreen inputs that I studied (section 3), but one outcome of this study was the importance of assigning those inputs to features judiciously, i.e. habits of the target user (in terms of interactions with applications) and usage that he will do of the interface has to be taken into account. The requirements of a control interface for Roombots have been specified (section 4).

Finally, I tried to define the specifications of a prototype of ideal interface to control the Roombots (section 5).

Then that I have construct a prototype of control interface for Roombots, I start to implement it. That is what I do in the second part of the project (part II).
Part II

Implementation of a QWidget to visualize Roombots

In this part, I will start the implementation of a QWidget to visualize Roombots. The implementation will be written in C++, and will use the Ogre3D library.

7 Introduction

The goal is to implement a QWidget that offer a 3-dimensional visualisation of the Roombots and the ability to interact with it. I start this part by defining libraries that I will used (Ogre3D: subsection 7.1, CEGUI: subsection 7.2 and Qt: subsection 7.3). Then I define the specifications of the QWidget I implemented (section 8), and after that I give some informations about the implementation (section 9) such as the structure of the code (subsection 9.1) and a presentation of some problems I encountered during the implementation process (subsection 9.3). Finally, I conclude this part of the project by an evaluation of the implementation (subsection 10.1) and by the elaboration of a list of further improvements that can be add to current version of the QWidget (subsection 10.2).

7.1 Introduction to Ogre3D

Ogre3D[16] is an open-source 3D rendering engine. In other words, it’s a library that gives you tools to create your own 3D scene. Main usage of Ogre3D is to implement video games, but it has a lot of domain of application. In fact, it is a nice solution for someone who have to create a 3D visualisation in his software and don’t want to spend it’s time on computer graphics.

7.2 Introduction to CEGUI

CEGUI[11] is a library providing windowing and widgets for graphics engines. In other words, it’s a toolbox to develop graphic interfaces containing windows and interaction tools (buttons, checkboxes, and so one).

7.3 Introduction to Qt

Qt[19] is an application framework that can be used for developing application software. One of the main uses of Qt is to implement a graphical user interfaces. This part of the Qt framework - the widget toolkit, allows the user to construct easily a graphic interface, by creating a windows and adding different graphic components in it. Such components are called "QWidget". Qt widget toolkit provides some basic QWidgets (such that sliders, text areas, labels, buttons, and so one) but it is also possible to create our own QWidget.
8 Specifications of the QWidget

The purpose of the QWidget is to provide a visualisation of a Roombots structure, and the ability to interact with it. So the user will be able to add modules in the visualisation and display it with or without a grid of connectors.

8.1 Selection of objects

In our visualisation, the user would be able to select an object. In the abstract representation - the representation of Roombots in our mind, we would like to select modules, active connectors (connector with ACM), and degrees of freedom.

Creating a visualisation that respect this will result in some constraints:

- I have to display a graphical representation of the degrees of freedom to provide the user the ability to select it. One solution will be to display the axis of the degree of freedom, but this would decrease the readability of the visualisation, and it is not efficient to select (with a mouse or on a touchscreen) an axis.

- How to select a connector that is connected to another?

For those reasons, I decided to construct another abstract representation:

- Modules will not be selectable.
- Connectors, both with and without ACM, will be selectable.
- Spheres (half module) will be selectable.
- Degrees of freedom will not be selectable.
- Central degree of freedom of a module will be represented by a cylinder, that is selectable.
- Connection between connectors will be represented by a cylinder, selecting the connection will result in selecting the active connector of the connection.
- Selected objects will be listed.
8.2 Interaction

They are three types of interaction the user can have with the Roombots visualisation:

The first one neither affects the set of selected objects nor the structure of the Roombots. For example, moving the camera is one interaction of this type.

The second one affects the set of selected objects but not the structure of the Roombots, this is for example a selection request. When the user orders a selection request on a selectable object that is not in the selected list, it will add it to it, and when he orders it on a selectable object that is in the selected list, it will remove it of it.

The third one doesn’t affect the list of selected object, but the structure of the Roombots. Two interactions of this type are "modify a degree of freedom" and "change the value of a ACM".
When he modifies the degree, each selected sphere will modify it’s degree of freedom to be set at the specified value, and each central degree of freedom associated to a selected cylinder will be set to the specified value.
When he changes the value of ACM, every active connector that is selected will modify the value of its ACM to the specified value.
Appearance and disappearance of connection between connector will be managed by the software, given the value of ACM, and proximity of other connectors.
9 Implementation

The code is written in C++. For the render engine, it has been decided to use Ogre3D as render engine. The choice has been done because Ogre3D usage is not really difficult to learn, and it is complete in terms of features. Another point is that Ogre3D is an open-source library. That can be really helpful in case of problems (you can check by yourself that implementation of what you use is as you expected).

For the widget toolkit, the first choice was to use CEGUI, because it is an open-source library too, and because it is the widget toolkit that recommend developers of Ogre3D. After trying to use it, the decision was taken to use Qt (subsection 9.3.1). Qt widget toolkit was chosen because it is one of the most widely used widget toolkit and then a lot of documentation over it can be found. Coding convention and user guide can be found in appendix (respectively B and A).

Documentation in html and latex has been generated by Doxygen[12].

The code is under GPLv3 license[29].
9.1 Structure of the code

**RBStructure class** is responsible of maintaining the concrete representation of the Roombots. In other words, it is a container of RBObject. RBObject is the common inheritance class for every object that is a component of Roombots.

![Inheritance graph for RBObject class](image)

**OgreWidget class** is the class that consists of the facade of the widget for Roombots. It contains five main attributes:

- *Attributes for Ogre* that are a scene manager, a camera, a render window, and other Ogre necessary components.
- *RBStructureManager* that is in charge of managing the Roombots structure.
- A *DegreeModifier* a dock widget to change the value of selected degrees.
- A *ConnectionModifier* a dock widget to change the value of selected ACMs.
- A *list of selected objects* a dock widget to inform the user about selected objects.

**MainWindow class** provides an example of usage of the OgreWidget class. It is composed of the OgreWidget, a dock to modify the camera position, and a menu bar.
9.2 Presentation of the implemented QWidget

The implemented QWidget is composed of four graphic components:

- **A visualisation area** that contains a graphic 3-dimensional representation of the Roombots structure, it is the main components of the QWidget.

- **A degree modifier widget** composed of a wheel to change the value of selected degrees, and a LCD to display the current value of selected degrees.

- **A connection modifier widget** composed of a slider to change the value of selected active connectors and a LCD to display the current value of selected active connectors.

- **A list of selected objects widget** that contains the set of selected objects. Objects are referenced by their name and a identification number.

**Characteristics of the visualisation:**
The visualisation contains all the objects of the structure. Each type of object has a specific color. Spheres are purple, connectors yellow, central degrees of freedom blue and connections red. When the object is not selected, the color is blade, and when the object is selected, the color is bright.

**Example of usage of the QWidget:**
To show what can be done using the QWidget, I implement a basic example. In this example, I create a window, in which I put the QWidget, a menu bar and a coordinate modifier to set the position of the camera.

![Figure 20: Screenshot of the QWidget in the example application.](image-url)
**Analyse of the QWidget:**

The QWidget allows the user to visualize a Roombots structure, on a grid or not, and to interact with it in two ways (selection not considered): change one degree of freedom of a module and change the value of an ACM.

The representation is a model, it looks not the same as the real roombots: connections and central degrees of freedom are displayed as cylinder, and modules are composed of two spheres.

It is possible to export the structure into a xml-file, but not to import it. I don’t implement the feature to import the structure from a xml-file due to lack of time. The implement would have taken me quite some time since I’ve no experience in parsing, I have therefore chosen to leave this feature aside so I could concentrate more on the quality of what I already had undertaken.

It is always the "end-part" of a module that turn when the user applies a change on a degree. This mean that when the user adds a module to the representation, and then change a degree of this module, it would always be the second part of the module that will move. This is what the user expect when the module has been added on a connector, but if it has not been added on a connector, determine which part of the module has to turn will require to know whether one of the two parts of the module is connected elsewhere. I suggest to solve this problem using a graph representing connection of objects in the further improvements subsection (subsection 10.2).

![Screenshot of the application](image)

(a) Four modules on a grid. (b) Six modules in a locomotion configuration.

Figure 21: Screenshots of the application
9.3 Problems encountered during implementation

9.3.1 Usage of CEGUI

Problem definition
At first, it was decided that I will use the CEGUI[11] library to manage interaction with the interface. Unfortunately, I encountered some problems trying to run the simple tutorial example to use Ogre and CEGUI together. We try to solve them with Stéphane Bonardi, but in fact we didn’t manage to make it works.

Option chosen
We didn’t solve it, because after discussion, we found that if we had problems to run the tutorial and didn’t understood why, it will be difficult to use it on different platform without having a lot of troubles. So we decided to come back to Qt, that is at least less bugged and more documented as CEGUI.
9.3.2 Ogre bounding boxes problem

Problem definition

To select object in the scene, Ogre use the concept of bounding boxes to determine which mesh has been selected. The concept is that every object in the scene is associated to a bounding box. When the user clicks on the scene with the mouse, a ray will be created, from the camera to the direction specified by the position where the click occurs. The user of the library can get an iterator over intersected bounding boxes. What people generally do is to take the first bounding boxes of the iterator and say that the selected object is the one associated to this bounding box. This seems to be a good idea but in fact, the bounding box associated to an object has to be of type AxisAlignedBox. This imply that the bounding box has to be a cuboid aligned on the global coordinate system. This is really problematic for two reasons. First if I consider a disk, changing the orientation of it will change the size of the bounding box. Second, if a small object is on the surface of a bigger object, trying to selecting the small object will result in selecting the bigger one, because the bounding box of the small will be include in the bounding box of the large. What people often do to resolve this problem is to construct a set of small bounding boxes instead of one large for large object (large mesh). But this solution is resource consuming (you have to construct a large amount of bounding boxes), and not completely reliable.

Figure 22: Ogre bounding boxes. How people generally solve the problem: define a lot of small boxes instead on one per mesh. Source: [32]

Option chosen

The chance I have here is that every component of Roombots is defined in terms of sphere or disk. The idea I developed is to use standard bounding boxes to get a set of potentially clicked object, and define between those, if one was really clicked. So when I have the set of potentially clicked object, I construct for each a sphere around it, and check if the ray intersect the sphere. The selected object is then the one for which the associated sphere intersect the ray at first.

51
10 Conclusion

10.1 Evaluation of the QWidget

10.1.1 Strengths

- The visualisation is intuitive, and I think it is not difficult to make the link between the graphic representation of Roombots and a real Roombots.

- The selection is precise, when the user clicks on an object, it is selected how he expects it.

10.1.2 Weaknesses

Remark: Here I only consider weaknesses of features that have already been implemented.

- The way the camera move is not efficient. The key point would be to define a way to select the point around which the camera turn.

- Instead of modify the value of a degree by setting the new value, it would maybe be a better option to set the modification of it, because relative movements are maybe easier to determine that absolute. Another advantage of this solution is that two different degrees can be modified of the same value, but each keep its offset.

- Modifiers (degree modifier and connection modifier) are placed out of the visualisation (in a dock). Putting them in the visualisation, would be valuable for the appealing of the interface. For example, when we select an object, corresponding modifier could open as a comics bubble located on the selected degree.
10.2 Further Improvements for the QWidget

In this section, I will try to determine what improvements could be add to the implementation of the QWidget to complete it.

- Implement an underlying directed graph to determine which part of a module has to turn when the user modifies a degree. The idea is to construct a graph in which each connectors and each degrees of freedom are represented by a node. One of them would be the root for each connected subgraph of the graph. If there is a grid in the subgraph, the root would be in the grid, otherwise, the way to select the root would be define. When a degree change occurs in a subgraph, first we can check that there are no cycles containing the degree changed, and second we determine which part has to move by finding the tree from the root to the node corresponding to the affected degree.

- Implement the connection with the real robot.

- Implement the loading of a structure from a XML file in the XMLManager class and add corresponding method in the RBStructureManager class.

- Implement an assembly mode to construct a structure using drag and drop.

- Implement a collision detection module.

- Port the code on a mobile device, use touchscreen and accelerometer inputs.
11 Project conclusion

In part I, I first talk about how to control modular robots in general (section 2) and after that I try to elaborate - step by step - specifications of an ideal interface to control Roombots (section 5). I emphasize the role that can play a good control interface for Roombots in the conclusion of this part (section 6), but I also highlight the fact that the implementation of an ideal interface as specified above will be a large project.

In part II, I start the implementation of a QWidget to visualize and interact with the Roombots. I define how will be implemented the widget and which libraries will be used (section 7), then I define the specifications of the implementation (section 8). I present the implementation, including the structure of the code, what problems I encountered and what the user will actually be able to do (section 9). Conclusion of this part of the project is essentially based on the evaluation of the implementation and further improvements that could be added to it (section 10).

One outcome of this project is the necessity of having tools to control modular robots, not only as an end goal, but also for the development process.

As I have already said before, I think that control tools are a key points of robotics in general and particularly modular robotics.

I see that implementing such tools is tricky, due to the large kind of structure that can take modular robots, but is feasible, and evolution of devices such that TabletPC and Smartphone offers a large field of investigation for developing control interfaces in general.
References

Texts


Softwares and libraries


   http://www.ogre3d.org/


   http://playerstage.sourceforge.net/

[19] Qt. Framework used for developing application software with graphical user
    interface.
   http://qt.nokia.com/

    http://www.quake3arena.com/

[21] Webots. Development environment used to model, program and simulate
    mobile robots.
    http://www.cyberbotics.com

Webpages

    http://www.apple.com/

    http://www.zath.co.uk/asphalt-5-hd-ipad-game-app-review/

    http://www.techno-science.net/?onglet=news&news=518

    http://avogadro.openmolecules.net/

[26] BIRG: Biologically Inspired Robotics Group, the former name of the
    Biorobotics Laboratory.
    http://biorob.epfl.ch/

    http://www.ros.org/wiki/simulator_gazebo

[28] Google: A corporation invested in internet and communication technolo-
    gies.
    http://en.wikipedia.org/wiki/Google

[29] GPLv3: A standard license for open-source software.
    http://www.gnu.org/licenses/gpl.html

    http://en.wikipedia.org/wiki/Impossible_object

[31] Inverse kinematics.
    http://en.wikipedia.org/wiki/Inverse_kinematics
http://axiom3d.net/forums/viewtopic.php?f=1&t=586&start=0

[33] Perceptive Pixel.
http://www.perceptivepixel.com/

[34] Qt documentation. http://doc.qt.nokia.com/

http://www.ni.com/labview/whatis/user-interface/f/

http://birg.epfl.ch/page68135.html

[37] Roombots: Source of the image.
source:http://biorob.epfl.ch/research/modular

[38] Towards an improved framework for YaMoR :Simulation and real-time control.
http://birg.epfl.ch/webdav/site/birg/shared/bluetooth/
downloads/report-final-summer.pdf

http://biorob.epfl.ch/yamor

[40] Youtube: video browsing site.
http://www.youtube.com/
A User Guide

A.1 List of components of the visualisation

Figure 23: Roombots Widget.

Figure 24: Visualisation of the Roombots widget.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Camera position modifier</td>
</tr>
<tr>
<td>2</td>
<td>Menu bar</td>
</tr>
<tr>
<td>3</td>
<td>Degree modifier</td>
</tr>
<tr>
<td>4</td>
<td>Connection modifier</td>
</tr>
<tr>
<td>5</td>
<td>List of selected object</td>
</tr>
<tr>
<td>6</td>
<td>Sphere (unselected)</td>
</tr>
<tr>
<td>7</td>
<td>Central degree of freedom (unselected)</td>
</tr>
<tr>
<td>8</td>
<td>Connection (selected)</td>
</tr>
<tr>
<td>9</td>
<td>Central degree of freedom (selected)</td>
</tr>
<tr>
<td>10</td>
<td>Sphere (selected)</td>
</tr>
<tr>
<td>11</td>
<td>Connector (unselected)</td>
</tr>
<tr>
<td>12</td>
<td>Connector (selected)</td>
</tr>
</tbody>
</table>
A.2 Description of components

**Camera position modifier** can be used to modify the position of the camera. The camera is oriented in the position of the center of the visualisation (0, 50, 0).

**Menu bar** is composed of two menus:

- **File menu** allows the user to *save structure into XML file* and *quit* the application.
- **Tools menu** allows the user to *Change background color* of the visualisation.

**Degree Modifier** allows the user to change the value of the degree associated to selected spheres and selected central degrees of freedom. It is composed of a lcd displayer that display the current value of the degree and a dial to change this value.

**Connection modifier** allows the user to change the value of the ACM associated to selected active connectors. When a connection is selected, the corresponding active connector is also connected.

**List of selected object** gives information to the user about what objects are selected.

**Sphere** is a graphic representation of a half module of the Roombots.

**Central degree of freedom** is a graphic representation of the degree of freedom between two halves of a Roombots module.

**Connector** is a graphic representation of a connector of the Roombots.
A.3 Interactions

A.3.1 Keyboard interactions

Keyboard can also be used to modify the position of the camera.

<table>
<thead>
<tr>
<th>Key pressed</th>
<th>Coordinate</th>
<th>Modification of the value</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>x</td>
<td>+10</td>
</tr>
<tr>
<td>A</td>
<td>x</td>
<td>-10</td>
</tr>
<tr>
<td>W</td>
<td>y</td>
<td>+10</td>
</tr>
<tr>
<td>S</td>
<td>y</td>
<td>-10</td>
</tr>
<tr>
<td>X</td>
<td>z</td>
<td>+10</td>
</tr>
<tr>
<td>Y</td>
<td>z</td>
<td>-10</td>
</tr>
</tbody>
</table>

A.3.2 Mouse interactions in the visualisation area

Left long-pressing movement will change the camera position. For example, if you move the cursor from the top to the bottom of the screen (and keep left-pressed), then the camera position will increase its height coordinate (y-coordinate). If you move from left to right, the x-coordinate of the camera will be decreased.

Left double-click is the input to select/unselect an object in the visualisation. If you double-click on an object that is unselected, it will become selected, if you double-click on an object that is selected, it will become unselected. If you double-click on nothing, nothing will append.

A.3.3 Mouse interactions out of the visualisation area

Interaction out of the visualisation is defined by standards of Qt. If you want any information about it, refer you to the documentation of Qt[34].
B Coding convention of the implementation of the QWidget for Roombots

B.1 Naming convention

B.1.1 Definition of patterns

Underscored: Words that composed the name is separate by an underscore (_), and is written in lower case. (ex: my_name)

Firstupper: The first character of every word that composed the name is upper case, the other are lower case. Words are paste. (ex: MyName)

Camelcase: Words that composed the name are paste, first start with a lower case letter, other words start with upper case, every non-starting letter is in lower case. (ex: myName)

Uppercase: Every letter is upper case, words are paste. (ex: MYNAME)

B.1.2 List of conventions

<table>
<thead>
<tr>
<th>Type</th>
<th>Pattern</th>
<th>Prefix</th>
<th>Suffix</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class</td>
<td>Firstupper</td>
<td></td>
<td></td>
<td>MyClass</td>
</tr>
<tr>
<td>Roombots Class</td>
<td>Firstupper</td>
<td>RB</td>
<td></td>
<td>RBMyClass</td>
</tr>
<tr>
<td>Macro</td>
<td>Uppercase</td>
<td></td>
<td></td>
<td>MYMACRO</td>
</tr>
<tr>
<td>Local variable</td>
<td>Underscore</td>
<td></td>
<td>my_variable</td>
<td></td>
</tr>
<tr>
<td>Class attribute</td>
<td>Underscore</td>
<td></td>
<td><em>my_attribute</em></td>
<td></td>
</tr>
<tr>
<td>Method parameter</td>
<td>Underscore</td>
<td></td>
<td>_my_parameter</td>
<td></td>
</tr>
<tr>
<td>Method</td>
<td>Camelcase</td>
<td></td>
<td></td>
<td>myMethod</td>
</tr>
<tr>
<td>Qt slot</td>
<td>Camelcase</td>
<td>SLOT_</td>
<td></td>
<td>SLOT_mySlot</td>
</tr>
<tr>
<td>Qt signal</td>
<td>Camelcase</td>
<td>SIGNAL_</td>
<td></td>
<td>SIGNAL_mySignal</td>
</tr>
</tbody>
</table>
B.2 Comment convention

B.2.1 Type of comment

File information : Multi-line, start with /*, end with */, each line of the comment start with *. Not used for documentation generation.

Brief description : One line, start with //!, used for documentation generation.

Elaborate description : Multi-line, start with /*!, end with */. Used for documentation generation.

Implementation comment : Help to understand the meaning of a block of statement. Either // if one line, or /* */ if multi-line. Not used for documentation generation.

B.2.2 Usage of comment

- Each file start with a file information comment.
- Each class contain a brief description and an elaborate description in its header (.h).
- Each method has an elaborate description in the source file(.cpp).
- Implementation comment are provided directly in the code when necessary.
C Schedule of the project

This project has been done in the scope of a semester project. Its duration was of 14 weeks. I spent approximately the same amount of time over each part of the project (part I and part II).

<table>
<thead>
<tr>
<th>Part</th>
<th>Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>Part I</td>
<td>Learning and applying of Mesoscale Modelling</td>
</tr>
<tr>
<td></td>
<td>Building of a RRT</td>
</tr>
<tr>
<td></td>
<td>Writing of the first part of the report</td>
</tr>
<tr>
<td></td>
<td>Finalizing the first part of the report</td>
</tr>
<tr>
<td></td>
<td>Writing of the second part of the report</td>
</tr>
<tr>
<td>Finalization</td>
<td>Submission of the final report</td>
</tr>
<tr>
<td></td>
<td>Preparing for the oral defense</td>
</tr>
<tr>
<td></td>
<td>Preparing for the oral defense</td>
</tr>
</tbody>
</table>

Figure 25: Schedule of the project