KIRC Project: Final Report

KIRC Team

January 20, 2014

Contents

1 Project proposal

1.1 Introduction

1.1.1 State of the art of robotic mapping

The ability of constructing a model of the spatial environment of a robot is one of the main challenges researchers face when trying to design fully autonomous robots. Current methods are often limited to static and finite environments[1]. Autonomous robots can be used for exploring an area not easily accessible by humans, or very large area where involving a human in the process would require too much effort.

1.1.2 Overview and objectives of the project

This project is named as Kinect Interactive Robot Companion (KIRC). It is a project done by Master1 students of the ENS Lyon in 2013/14 academic year for the course 'Integrated Project'. The goal of this project is to build a robot able to move autonomously while providing a real-time 3D map of the environment seen so far -along with a video stream- through a website. Moreover, an admin user with more privileges should be able to give commands to the robot through an interface. We resticted ourself to the explorations in a building (or in a environment where the ground is similar). We chose this because dealing with natural obstacles (water, rocks, etc.) can be problematic and is an entire problem far away of our competences.

We can look at the benefits of this project from different perspectives. One of these, which has to be pointed out, is that this project has a direct impact on our experience, which is also a goal of the course, to make students familiar with an international type of working environment, since the project group is formed by people from different countries.

1.2 Structure of this document

In this document we will describe all the details of the project starting from the first to the final stage. By the next subsection we will describe the system architecture, the work package division and the relation between the work packages. We will conclude this chapter by presenting the group members and the time/human analysis. We will also look at the expected outcome of the project by the end.

Since the project followed a work package division, in the next chapters we will look at each work package in detail. Starting from their internal goals, that will merge to other work packages and form the final result, to their resolution methods. We will also describe what difficulties have been faced and how the team dealt with them.

Chapter two will describe the *Hardware* part in detail, followed by the *Artificial Intelligence (AI)* in chapter three. This is a logical flow since the Hardware is directly used by the AI part which, through the use of an API, will drive the robot as presented in chapter three.

The *Mapping* work package follows in the fourth chapter describing in detail how the robot constructs the map using Kinect (we will see the Kinect in detail in the hardware part). Moreover, the robot should be able to be accessed and controlled remotely, which is among the goals of the *Connectivity* work package. We will see this in the fifth chapter and the document will conclude by describing some future works that can be done to improve this project.

1.3 System Architecture

The following diagram shows the current system architecture.

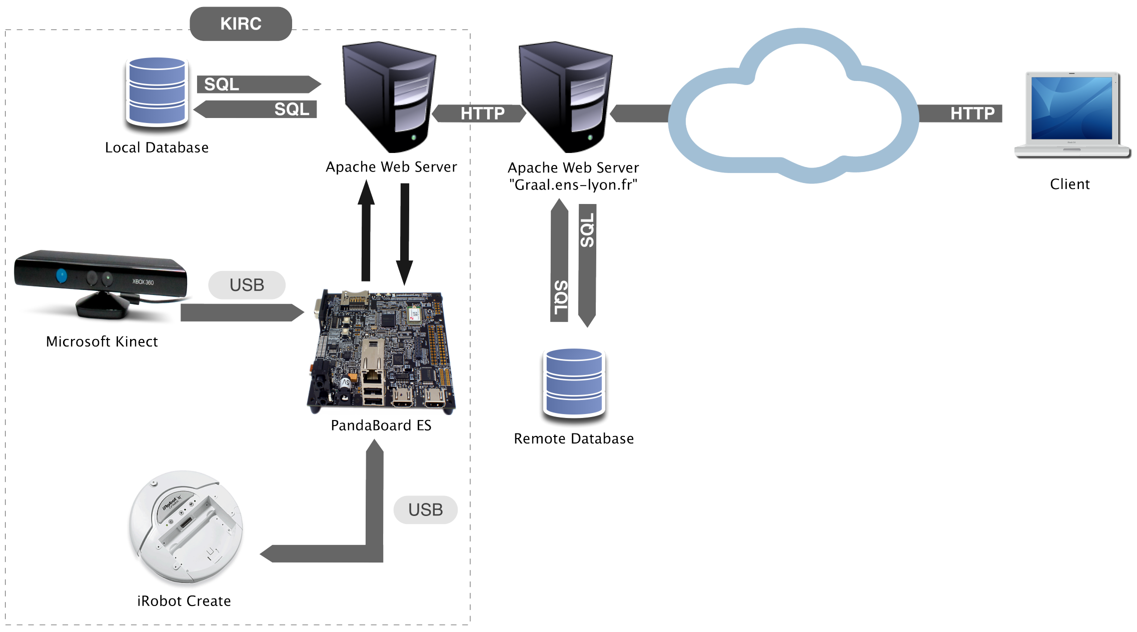


Figure 1: System Architecture Diagram

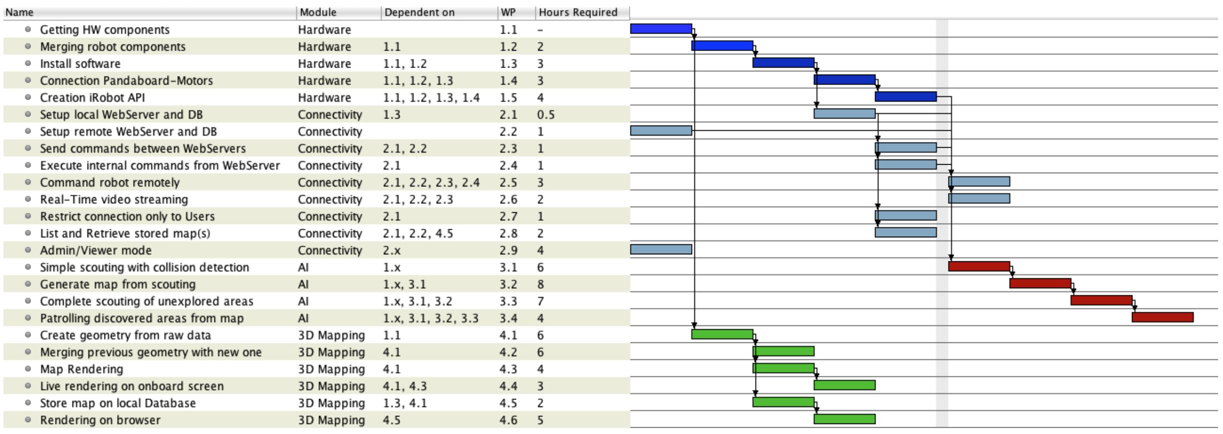
The section named "KIRC" represents the structure of the robot itself.
Below a description for each of the main components of the architecture.

- *PandaBoard ES:* the main component of the robot. It connects with the other components and this is where all the main scripts are executed.
- *iRobot Create:* the component which is responsible for the movement of the robot. Through an API, the PandaBoard can interact and control this component for moving.
- *Microsoft Kinect:* the device which allows the robot to 'see' its surroundings. The data from this component is used for AI, Mapping and video streaming.
- *Apache Web Server:* the web server used for ingoing and outgoing communications. It communicates with the local Database and the remote web server (located on "Graal.ens-lyon.fr").
- *Local Database:* Database where the mapping data is stored. It replicates onto the Remote Database, for synchronization.
- *Apache Web Server "Graal.ens-lyon.fr":* the remote web server which allows the Local Webserver to be contacted from outside the ENS network. It also contains the public website of the project.
- *Remote Database:* Database containing user logins and the mapping data retrieved from the Local Database with the replication.
- *Client:* whoever connects to the Apache Web Server "Graal.ens-lyon.fr" and has an account (either viewer or admin) for accessing the functionalities of the robot.

The current System Architecture takes in consideration previous successful embedded projects as well as experience acquired during the study of the project. In fact, the overall System Architecture went through several changes, although small, mainly due to the need to adapt to the current available resources (network, components, processing power, etc.).

1.4 Work Packages (WP) and their relations

The table below shows the tasks list in each work package with the time planned for each task and the dependency relationship.



Name	Module	Dependent on	WP	Hours Required
Getting HW components	Hardware		1.1	-
Merging robot components	Hardware	1.1	1.2	2
Install software	Hardware	1.1, 1.2	1.3	3
Connection Pandaboard-Motors	Hardware	1.1, 1.2, 1.3	1.4	3
Creation iRobot API	Hardware	1.1, 1.2, 1.3, 1.4	1.5	4
Setup local WebServer and DB	Connectivity	1.3	2.1	0.5
Setup remote WebServer and DB	Connectivity		2.2	1
Send commands between WebServers	Connectivity	2.1, 2.2	2.3	1
Execute internal commands from WebServer	Connectivity	2.1	2.4	1
Command robot remotely	Connectivity	2.1, 2.2, 2.3, 2.4	2.5	3
Real-Time video streaming	Connectivity	2.1, 2.2, 2.3	2.6	2
Restrict connection only to Users	Connectivity	2.1	2.7	1
List and Retrieve stored map(s)	Connectivity	2.1, 2.2, 4.5	2.8	2
Admin/Viewer mode	Connectivity	2.x	2.9	4
Simple scouting with collision detection	AI	1.x	3.1	6
Generate map from scouting	AI	1.x, 3.1	3.2	8
Complete scouting of unexplored areas	AI	1.x, 3.1, 3.2	3.3	7
Patrolling discovered areas from map	AI	1.x, 3.1, 3.2, 3.3	3.4	4
Create geometry from raw data	3D Mapping	1.1	4.1	6
Merging previous geometry with new one	3D Mapping	4.1	4.2	6
Map Rendering	3D Mapping	4.1	4.3	4
Live rendering on onboard screen	3D Mapping	4.1, 4.3	4.4	3
Store map on local Database	3D Mapping	1.3, 4.1	4.5	2
Rendering on browser	3D Mapping	4.5	4.6	5

Figure 2: Work Packages Diagram

As we have seen in the previous sub section the project contains different parts. Since the project was done in group we needed to divide the tasks in work packages. We divided the total project into four work packages. These are Hardware, Artificial Intelligence (AI), Mapping and Connectivity. This document presents each working package chapter by chapter, but now, this section will deliver an overview of the main tasks in these packages and how they are related.

The Hardware work package is the first. It includes getting all the hardware components and configuring them, but also delivering a usable specification for other modules (like AI - controling the robot). Being given the natural flow between hardware and the commands used for controling it, Hardware work package also provides an interface for the Artificial Intelligence (AI), with full command specifications. The Artificial Intelligence is responsible for moving the robot in a productive way.

The main goal of the movement of robot is to see its environment and produce a map, which is done by the Mapping work package. This work package uses the data from the Kinect, creates a useful map of its environment and stores it in the local database server. More details of this in chapter four.

After producing the map, it has to be presented to the user in a convenient way. The Connectivity work package will retrieve the data from the local server and show a graphical representation of the data. This and other features of the Connectivity work package are discussed in detail in chapter five.

In the next subsection we will see how the members of the group are divided and assigned tasks, what are the deliverable documents and the expected output for the project.

1.5 Team members, Deliverables and Expected outputs

In this section, we will discuss the internal role subdivision, the deliverables and the expected outputs from this project.

1.5.1 Team members

This project group is composed of seven Computer Science Master 1 ENS-Lyon students. The group is divided as follows:

Balthazar Bauer QA manager, Developer

Guillaume Cordonnier GUI supervisor, Documentation supervisor, Developer

Julien Le Maire Developer

Mihai Popescu Developer, AI supervisor

Romé Tillier Communication responsible, Hardware responsible, Developer

Alberto Vaccari Project manager, System architect, Developer

Amir Wonjiga Hardware responsible, Developer

Based on this, Alberto Vaccari is mainly working on Connectivity, Julien Le Maire and Guillaume Cordonnier are working on the Mapping and the rest of the group members are working on Hardware and Artificial Intelligence work packages.

1.5.2 Deliverables and Expected outputs

Since this is a course project it is expected to submit some documents. The submission of *project proposal* was expected at first time. The *mid-term report* presented the progress of the project. Both the project proposal and mid-term report have been submitted. This document is the final report which describes all things related to the project and future works that need to be done to improve this work.

At the end of this project, will be presented: a robot that detects its environment and produces a useful map, a way of presenting this map to users, wherever they are located on the internet, a real time video streaming for the users and a way for an admin user to give orders (commands) to the robot from a remote server.

In the next chapters we will see in detail how each of the working packages are implemented, what difficulties we faced and how we come up to the solutions.

2 Hardware

2.1 Initial goals

The main objective of this work package is to identify, analyze, obtain and put together all the components of the robot, thus building and obtaining KIRC's final structure. This thing assumes 3 major distinct parts of the hardware system: PandaBoard, iRobot Create platform and Microsoft Kinect. Aspects such as communication and power supplying had to be taken care of. This is necessary for ensuring the good interactivity between parts, hence a successful merging operation for them.

2.2 The progression throughout the project lifespan

The first thing that had to be done was to identify all the needed hardware components. These were analyzed and, when available, multiple possible solutions were discussed, thus ensuring the best option. Here, two important aspects, which had to be takent care of, are the communication between the main parts (PB, iRobot and Kinect) and their power supplying methods. Since the robot had to be movement independent, all the major components needed independent power source.

For moving the robot, the best option was iRobot since it is delivering absolutely everything we need for exploring any area we need. And this is what we were looking for when we first decided the goals and objectives of the project. Moreover, iRobot is offering plenty of sensors, which can give useful information regarding its enviroment (such as wall sensors - for detecting walls or obstacles - three bump sensors - for detecting any touch of an object on its left, right or front side - infra-red receiver, cliff sensors - for detecting any edge near the robot - and so on ...). One view of the robot, with its main aspects, in figure 3.

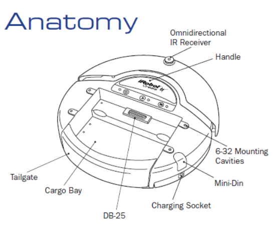


Figure 3: iRobot Create Platform

PandaBoard delivers a lot of technical advantages and functionalities, the most important of them being the WiFi connection, which we had to use in order to make our robot independent in movement. An overview of its specification can be seen in figure 4.

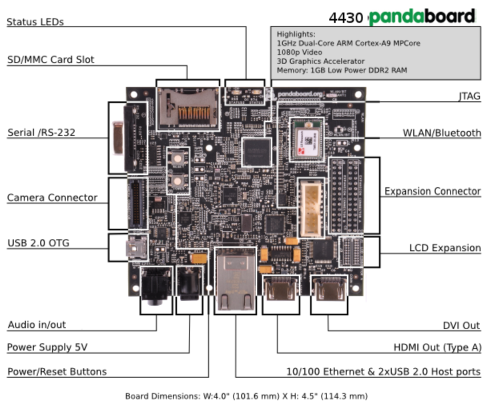


Figure 4: PandaBoard specification

Moreover, comparisons with other (products and) providers were done, so shops and prices were established for every part of the robot. As a result, the Hardware Proposal was finished. In the following period, components were obtained step-by-step, hence the merging began.

Once the iRobot Create platform was received, it was tested independently, but also with PandaBoard on it. PandaBoard was using too much current for the 5V Duracell battery Power Pack and sometimes, on high peaks, it was suddenly shutting down. So anotehr Power Pack had to be purchased, and this resolved the problem with power supplying for the board.

2.3 Difficulties

Since the robot had to move independently, it had to have its own independent power source. Thus, one important problem that had to be faced was the power supplying method for each of the major components of the system, especially for MS Kinect and PandaBoard.

For solving the power supply problem, multiple options were taken into account. For both Microsoft Kinect and PandaBoard, one could use a big independent battery, but this option was very tricky because of the configuration that had to be made in order to work properly. Another solution was to use two batteries, but the compilicated thing was that extra work was needed on cutting cables and making new circuits. One possible scenario was to use the voltage from iRobot, to power MS Kinect and/or PandaBoard. The problem here was that iRobot's battery would have suffered a lot from this. Finally we reached the conclusion that the best option is to use Power Packs for powering PandaBoard and a set of batteries for MS Kinect, for which a tutorial has been folowed ([13]).

2.4 Technical details

Let us see some basic commands for iRobot, presented as an API specification. You can find the whole documentation on our project website [8].

```
 int startOI (char * serial)
```

Opens the specified serial port to the Create and starts the Open Interface. The Create will be started in Passive Mode. This command or startOI MT must be called first before any others. Parameters: serial The location of the serial port connected to the Create Returns: 0 on success or -1 on error

```
int stopOI ()
```

Stops the Create and closes the serial connection to it. The connection can be restarted by calling startOI. Returns: 0 on success or -1 on error

```
int drive (short vel, short rad)
```

Drives the Create with the given velocity (in mm/s) and turning radius (mm). The velocity ranges from -500 to 500mm/s, with negative velocities driving the Create backward. The radius ranges from from -2000 to 2000mm, with positive radii turning the Create left and negative radii turning it right. A radius of -1 spins the Create in place clockwise and 1 spins it counter-clockwise. A radius of 0 will make it drive straight. Parameters: vel The velocity, in mm/s, of the robot rad The turning radius, in mm, from the center of the turning circle to the center of the Create Returns: 0 on success or -1 on error

```
int turn (short vel, short rad, int angle, int interrupt)
```

Moves the Create at the specified velocity and turning radius until the specified angle is reached, at which point it will stop. If interrupt is non-zero, the Create will stop if it detects a possible collision (bump, cliff, overcurrent, or wheel drop sensors are activated). Velocity ranges from -500 to 500mm/s (negative values move backward) and turning radius ranges from -2000 to 2000mm (negative values turn right; positive values turn left). A radius of -1 will spin the Create in place clockwise and a radius of 1 will spin it counter-clockwise. A radius of 0 will drive straight. Be careful to check your inputs so that the Create does not get stuck in this function. For example, make sure you do not tell the Create to drive straight with this function. Parameters: vel Desired velocity in mm/s rad Desired turning radius in mm angle Angle in degrees the Create should turn before stopping interrupt Set to 0 to have the Create ignore collisions or non-zero to have it stop on collision Returns: Angle turned or INT MIN on error

3 Artifical Intelligence (AI) work package

3.1 Initial goals

After having all the hardware there must be some program or algorithm who will decide where to go. Because the hardware by itself cannot make decisions. For this reason we need the AI work package who will take the robot through the environment by calling functions provided by the API. The objective of the AI work package is to assure an efficient movement of the robot. The Robot should have to make the movement without difficulties even if the mapping work fails with some error. The exact moving/making the wheel to rotate is not the aim of AI which is implemented in the hardware part. The AI will decide which way to take based on the algorithm running. There are different algorithms to make exploration.

In this chapter first we will see what API is used to access the functionalities of the iRobot, what algorithms do we use and why we choose those algorithms. It will also be described how we came up with the method that was used for implementing more than one algorithm, in order to get the benefits from each algorithm. Finally, we will finish by specifying what difficulties we face and how we solved those difficulties so that the desired output was obtained.

3.2 The progression throughout the project lifespan

3.2.1 Application Program Interface for the iRobot

As described in the hardware part, iRobot Create is an affordable mobile robot platform for educators, students and developers. Different peoples/ groups of programmers develop an API for iRobot. From this we can find PyRobot[2], oisimple C interface[3],libBiscuit[4], libcreateoi[5] and more others. From this APIs we choose libcreateoi (Create Open Interface Library (COIL)) since it is implemented in C and it contains all the functionalities provided by the iRobot. Though, for some commands, we had to implement and use our own API, for better accuracy in testing.

In the next subsection we will see the algorithms we use and how we merge two algorithms to maximize the accomplishment of our goal.

3.2.2 Implemented Algorithms

Our main goal while implementing these algorithms is to explore as much as possible areas in minimum time. There are different types of exploration algorithms, in this section we only see those which are used for our goal. You can refer to the WiKi page[6] for more information about exploration algorithms. The very naive algorithm is Random walk. It is simply to proceed in a straight line until a junction is reached, and then to make a random decision about the next direction to follow. Although such a method eventually may find the right solution, this algorithm can be extremely slow.

The first naive algorithm is a random walk: the robot simply proceeds in strait line until a wall in found. Then it arbitrarily decides to turn left or right with equal probability, and starts again from the beginning. While this algorithm seems really simple and efficient, tests on the simulator have shown that in some basic case, the robot made a infinite loop on a very restricted area, as you can see in Figure 5.

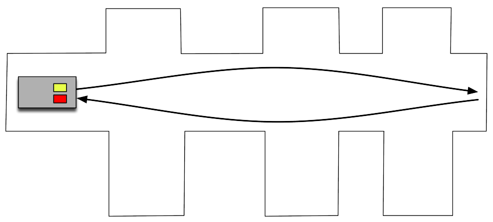


Figure 5: Example of infinite loop with a basic random walk algorithm

Thus we have decided to slightly modify the algorithm:

```
random_walk2:
While True
    Let X a exponential random variable and t a time variable
    While (NO_WALL and (t < X))
        Go_Straight
        update t
    end while
    Turn left with probability 1/2 and right with probability 1/2
end while
```

The other algorithm is a wall following algorithm. As the name indicates this algorithm will follow a wall from one side, left or right. This algorithm is the well known maze solving algorithm. But the limitation on this algorithm is the case when we have a loop, as we can see in figure 6

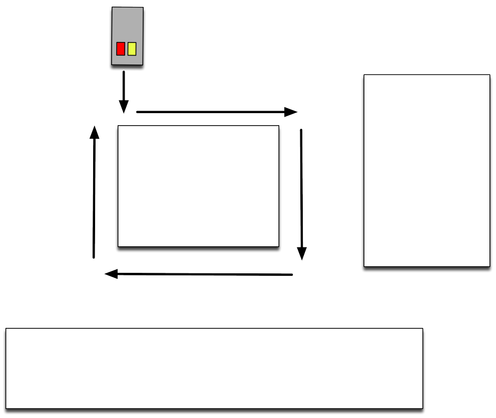


Figure 6: The wall follow algorithm in a loop

As this algorithm just follows wall, if the wall is like as shown in the figure or the same like that having a loop the algorithm will get into infinite loop. Here it come the pledge algorithm which is named by the creator of the algorithm (a little boy of twelve, who was called John Pledge). This algorithm solves the above problem by counting the turns. Go straight until it finds an obstacle. When it finds an obstacle turn to the left (right) and start counting. Whenever you turn to the left decries and when you turn to the right increase the counter (vice versa). When the counter is 360/-360 it assumes that we encounter a loop so he will go straight and restart the whole execution again.

Even if this algorithm solves the problem of maze our goal is to make an exploration of maximum area in the minimum possible time. The drawback of this algorithm is, it cannot explore all the areas. To solve this we merged the random walk algorithm with the pledge based on some time. The merged algorithm runs like this, assume A1 and A2 are pledge and random walk in respectively,

```
begin_time= getTime()
now=getTime()
    int N=30
      while true
          While (now - begin_time) < N
                {
                  begin_time= getTime()
                  apply A1
                   now=getTime()
                  }
           While ((now - begin_time) < 2*N )
                  {
                  begin_time= getTime()
                  apply A2
                   now=getTime()
                  }
          N = N*2
```

So using this method we implemented an exploration algorithm which has a better coverage than the pledge. We have decided to test some of our algorithms on few simulators. First we have constructed our own simulator (in C++), but this simulator was slow (for big maze), and not so beautiful, so we have found a simulator named micromouse simulator[7] very smart (The algorithm should be made in python). It is important to notice that both simulators used was "discrete simulator", so was far away of the reality, it is why, we have no assurance after the tests made that our algorithms will be okay for the real world.

3.3 Difficulties

After starting to test the algorithms in practice, we faced different types of difficulties. From these, we will describe the main ones in this section. The first challenge was how to find an algorithm that will give a better output for our goal. As we have showen above, we merge two different algorithms to get an algorithm which is based on time.

The other main serious problem was the hardware's errors. The AI part is communicating with the iRobot, getting data from it and making decisions using those data. But when the iRobot implements the commands it is not exact or perfect. i.e there are some errors and they were incrementing as the running time increases. So when trying to get, for example, the distance it traveled, we found unpleasant errors. This was the same for the angles that it turns. We made a lot of tests and approximation (based on computations) to make the errors as small as possible.

4 Mapping

4.1 Initial goals

The point of the mapping work package was to capture the geometry of the area surrounding the robot through the Kinect and to translate it to a more understandable model. This model should be usable either by the AI to find its path in the map or to complete unexplored areas, either to a remote client to draw a 3D representation.

Although this have been done before, the difficulty here is that we want the map to be created in real time by the robot itself.

4.2 The progression throughout the project lifespan

We began by the robot side, creating an interface with the Kinect using the library *libfreenect*. The output of the Kinect (Fig 7 below) consist of two images: one regular image which is a grid of RGB pixels, and a second image which gives for each pixel of the first image it's distance relatively to the Kinect sensor.

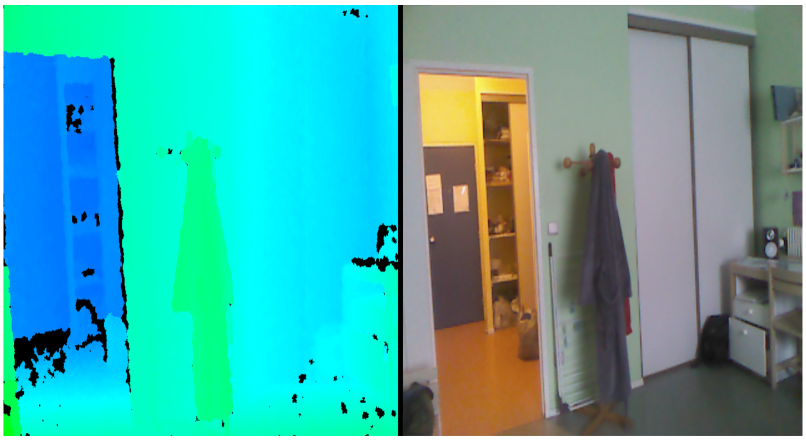


Figure 7: Raw output of the Kinect

Then, from the Kinect output we created a point cloud (Fig 8 below), by converting pixel positions and their depth to a set of 3D points. We planed to use a library called *Point Cloud Library* to convert this point cloud to polygons and to merge it into to the global point cloud by comparing interests points and finding the right alignment of the point cloud.

This is a standard technique used in 3D reconstruction, and one of the greatest benefit of using it is that it would have allowed us to extract the exact position of the robot from the alignment of the two points clouds and an approximate position of the robot, thus constantly correcting any difference between the real position of the robot and the position that the AI calculate when the robot moves.

However, theses two points where found to be to slow to be usable on the Pandaboard: it took more than 10 minutes to convert the point cloud into polygons -and a lot more to perform the alignment- on a computer with a 2Gz intel i5, so probably more on our little ARM Cortex-A9 embedded on the Pandaboard. This delay cannot considered as real-time reconstruction, so we switched to another solution. In this new solution, we must rely on the information from the hardware and AI part to have the position of the robot. This is very approximate and should be improved in a future version.

Thus, we decided to forget about the *Point Cloud Library* which was performing most of the steps of our reconstruction algorithm, and to create our own algorithm. The idea is to split the space in small aligned cubes of constant size that we call *voxels*. A voxel is considered full when a certain density of point of the point cloud is into it. In our model, a voxel can be either a full block or an empty block, but it can not be partially full. We used two methods for considering the density of points in a voxel, and a comparison of it can be found in the section 4.3.

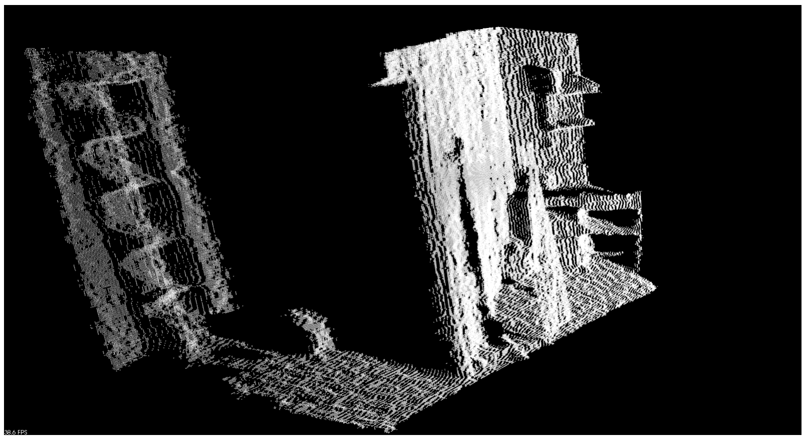


Figure 8: 3D point cloud created from the depth map of the Kinect

The advantages of this representation is that the precision can easily adapted to the resources available by changing the size of the voxels. It can also easily be used by the AI part: the robot can test to see if there is an obstacle at some distance of it, or if he is free to go.

Moreover, displaying this matrix of voxel is easy: we can then just show the activated voxels to have a good representation of the geometry. We have implemented a Web application for rendering the maps, using WebGL and a javascript library named three.js. By using theses technologies, we allow virtually any platform to use them, and then the resulting map can be explored by any web browser.

The figure 9 show the result rendered in a web browser. We used a technique for rendering the voxels that make them looks less "cubic", by cutting the isolated edges.

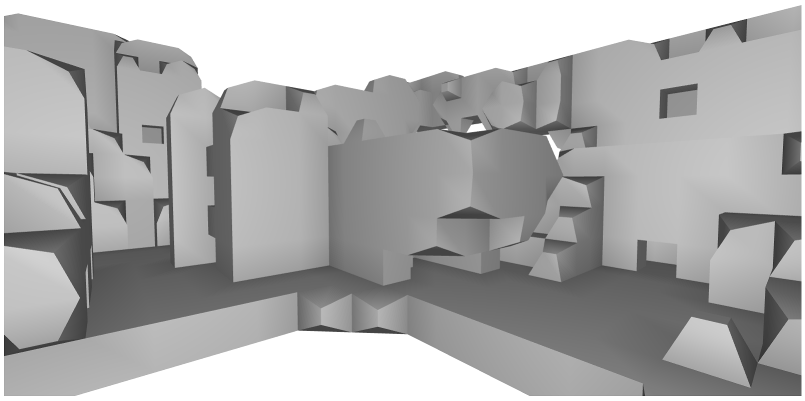


Figure 9: Example of a map under construction rendered into a web browser

4.3 Different methods for the density of points into a voxels

We used to different techniques for updating the density of points into a voxel. In this section, we will describe them, along with some ideas for improving them.

4.3.1 Basic density

This method is the most basic possible: whenever we have a point in a voxel, we add 1 to the density of this voxel. When this density reach a certain threshold, the voxel become full. In our current implementation, the threshold depend on the time when the first point have been added into the voxel. This method is very fast and provide pretty good results. The figure 10 show the result of this heuristic when the robot don't move.

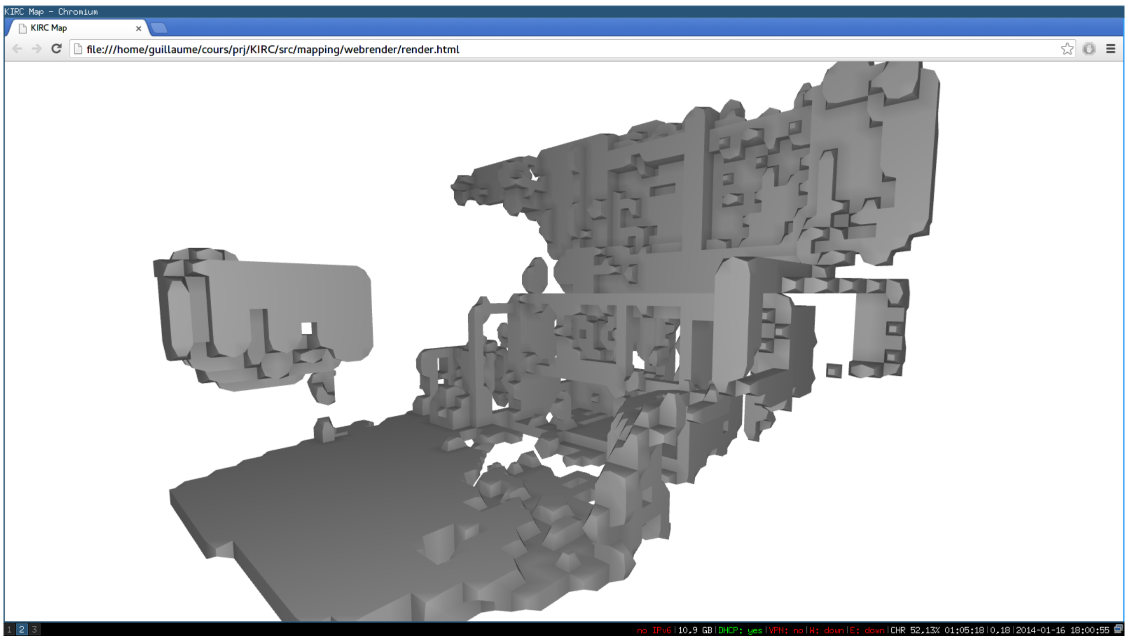

KIRC Map - Chromium
KIRC Map
file:///home/guillaume/cours/prj/KIRC/src/mapping/webrender/render.html
1 2 3
no IPv6|10,9 GB|DHCP: yes|VPN: no|W: down|E: down|CHR 52,13% 01:05:18|0,18|2014-01-16 18:00:55

Figure 10: Image of a room with shelfs with the Basic density method and a fixed Kinect

4.3.2 Density with negative points

With this method, we use the information that seeing a point means that there is nothing between the point and the Kinect. Thus, for each point we see, We add it to the corresponding voxel with a weight of 1, and then we construct a virtual ray between it and the Kinect, and add negative points (with weight -1) at regular intervals -typically the size of a voxel- along this ray.
A variation of it can be to add smaller negative weights -like (1 / (distance of the point)) - to make sure that any obstacle can be seen on the map.

This allow to reduce the noise seen between two images (because the point seen too close to the image are compensated by the the point that are seen too close).
But the main advantage of this method should be (we haven't properly tested it) the idea that the shapes left on the map by dynamics object should disappear: with the first method, whenever something was moving in from of the Kinect it's image was definitively added to the map. With this method, it is removed after a while because the virtual ray goes through the remaining image.

The main drawback of this method is that it is more expensive in terms of calculations. Moreover, on our test (and it can be seen on the provided images) we notice that the models tend to be less representatives of the reality, even if the resulting map do have less noise. For example, one the figure 10, the wall and the floor have more noise than on the figure 11 but the shelf can be recognized really more easily.

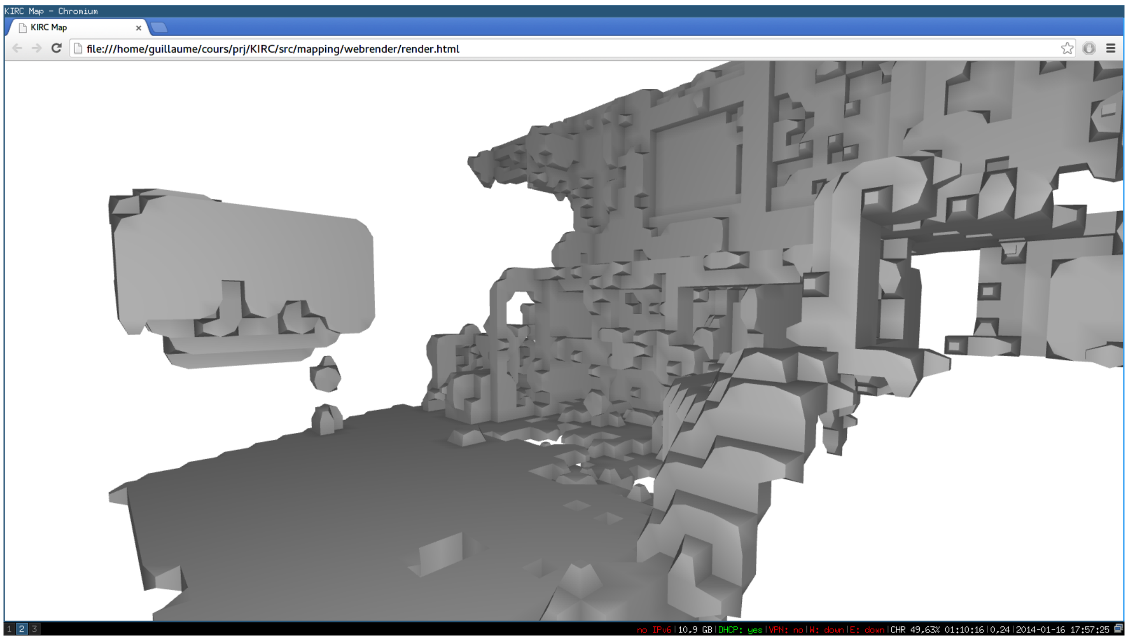

KIRC Map - Chromium
KIRC Map
file:///home/guillaume/cours/prj/KIRC/src/mapping/webrender/render.html
1 2 3
no IPv6 | 10,9 GB | DHCP: yes | VPN: no | W: down | E: down | CHR 49,63% 01:10:16 | 0,24 | 2014-01-16 17:57:25

Figure 11: Image of a room with shelfs with the Negative density method and a fixed kinect

4.3.3 Ideas for improving both methods

We believe that one idea for improving theses methods can be to take into acount the fact that some points are added into a voxel even if they are almost exactly between two voxels. For example, if a point is on the limit of two voxel, we can add half a point in each. Extending this idea, each point can be spited into several voxels if they are close enough, with the weight of each part depending on the distance between the point and the voxel. This idea can help drastically reduce the noise, and for negative weights, it can help the diffusion of the weights along the virtual ray.

5 Connectivity

5.1 Initial goals

The Connectivity work package has the main goal to provide remote features to the robot; such as remote access/control, live video streaming and data synchronization. The work packages also includes the setup of both a local web server (running on KIRC) and a remote web server (available at http://graal.ens-lyon.fr/kirc); these web server are used for communicating with each other and transferring map data.

5.2 The progression throughout the project lifespan

The progression of the Connectivity work package throughout the project was split into different parts, mainly due to the not immediate availability of all the components. The first part focused mainly on the remote web server, with an initial setup of the DB and the website, the management of Users and Sessions, creation of an internal communication system between team members, remote execution of commands directly on the robot and the live streaming from Kinect.

The following section concerned the board for the robot, with its installation and the connection between its local web server with the already functional remote web server.

The last section focused on completing the remote access and remote control of the robot, this includes controlling the movements of the robot remotely when in admin mode (through a visual interface on a browser), viewing the live stream of the robot and managing the previously stored maps.

5.3 The achieved goals

Although some delays, mainly due to the unavailability of certain components, all the initial goals have been achieved. It should be noted that the current proposed solution contains several changes from the initial soltion, which have been required due to the internal structure of the ENS network. It was not possible to directly communicate to the robot from the remote webserver, as well as not being able to directly insert data on the remote database.

5.4 Technical details

The approach it has been chosen for the communication between a remote User and the robot is to allow the User to connect to a remote web server, which can communicate with the robot locally. This is done by running a further webserver on the robot itself; in this way it is possible to create a simple API con the intercommunication as most of the security issues are handled by the remote server.

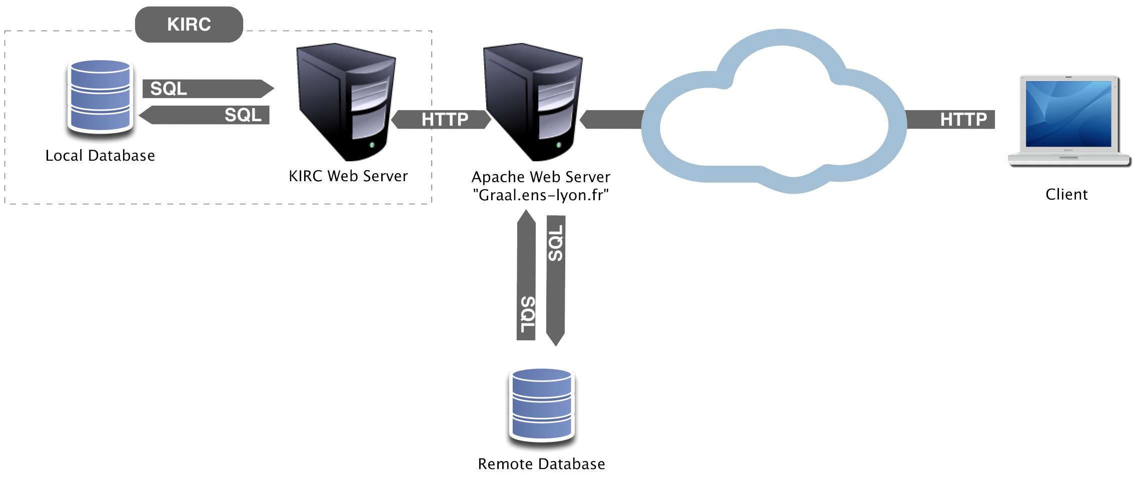
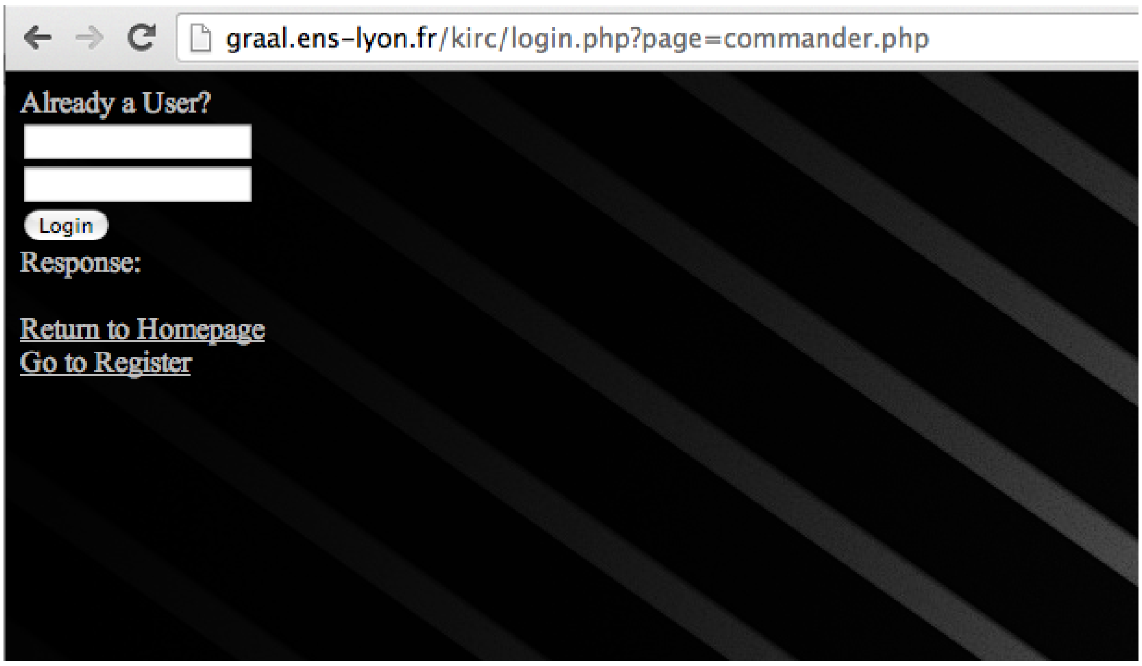


Figure 12: Connectivity Diagram

5.4.1 Robot Commander Page

The User is able to connect to a web interface after having successfully logged (figure 13).


graal.ens-lyon.fr/kirc/login.php?page=commander.php

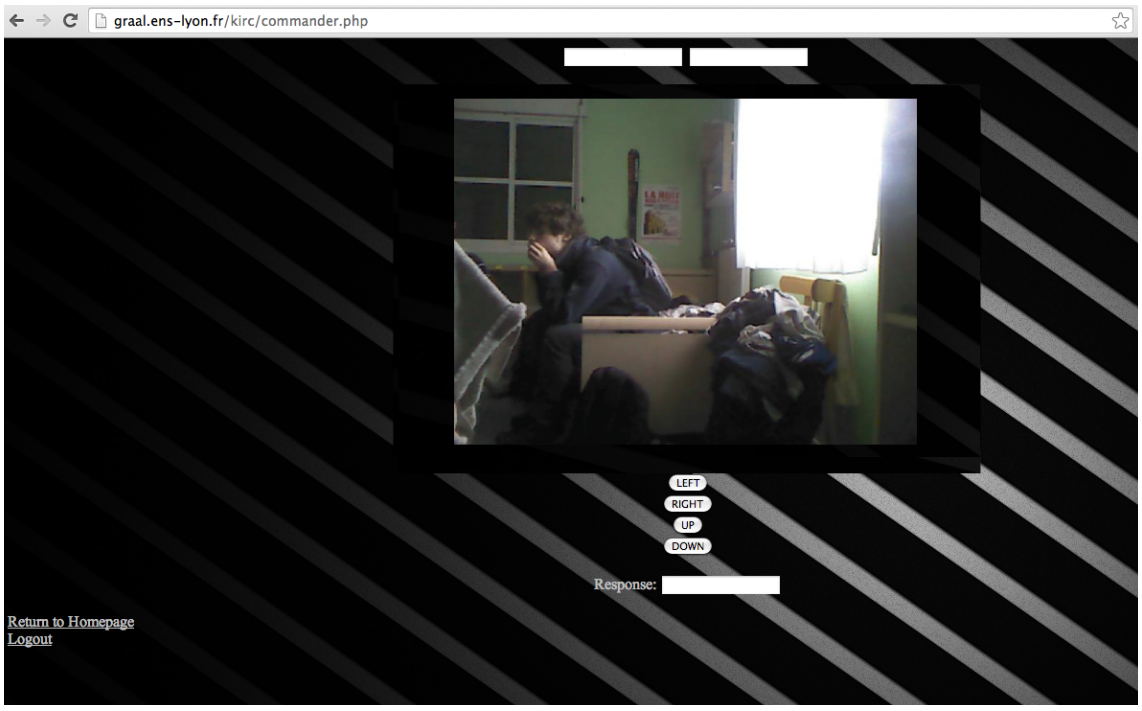
Already a User?

Login

Response:

Return to Homepage
Go to Register

Figure 13: Login page


graal.ens-lyon.fr/kirc/commander.php

LEFT
RIGHT
UP
DOWN

Response:

Return to Homepage
Logout

Figure 14: Commander page

Using the commander page (figure 14), the User is then able to see the live stream (which is a secondary RGB video stream from the Kinect, created using Intrael[12] from the robot and, if an admin, override the AI and control the robot.

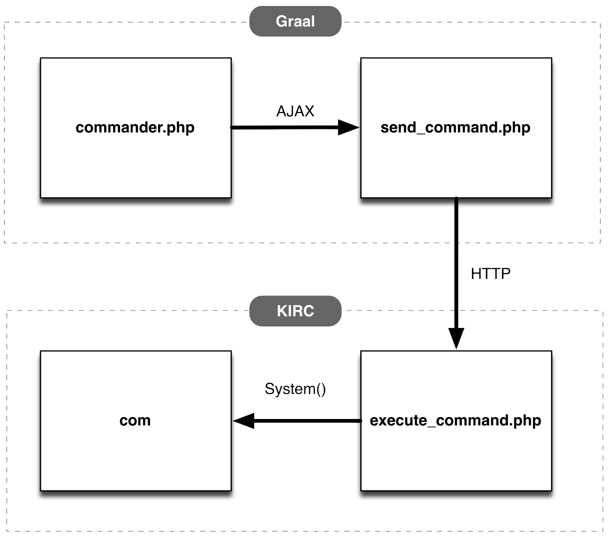


Figure 15: Execute command diagram

As represented in figure 15, whenever a button is pressed (for instance for moving the robot forward) the command is sent with an AJAX request from the `commander.php` to the page `send_command.php`, which handle the request, by checking its integrity and correctness and then forward it to the page `execute_command.php` located on robot web server. This page will check one more time for integrity and then execute the command through the script `com`, which communicates directly with the motors.

It should be noted that the extra integrity check on the robot side is performed mainly for adding a layer of protection to this rather unsafe solution. Some security is added by the fact that for being able to successfully skip any authentication and execute movement commands to the robot a prior knowledge of the IP of the robot, the exact script page and the exact request content are needed. Some variable names change as well through this process making it even harder to be accessed by a curious person. In conclusion, the solution has been seen as sufficient for the current context; further layers of security can be added in future versions.

For further reference, visit the homepage of the project[7].
For a complete description of the scripts and pages in the Connectivity work package, please refer to the textfiles available in the Git repository of the project[11].

5.5 Difficulties

This work package posed a lot of challenges which were mostly overcome through research or prior knowledge. The main difficulty was regarding the need of using a remote webserver without having full admin rights; in addition, the web server being on the DMZ (Demilitarized Zone) of the ENS network forced us to research even further and apply workarounds to being able to achieve our goals. The imposed limitations caused a lot of delays and double-work (in fact, the Connectivity module has solutions for both a standard network and an ENS-like network).

6 Conclusion

One of the main objectives was to contribute to the several communities working on the OpenKinect project as well as on facial recognition and 3D mapping; unfortunately this was not completed. However, the other objective of this project was to be able to construct a robot able to map it's environment autonomously using the Kinect, while creating a map of the explored area along with a video stream from it's current position. Our robot now can create 3D model in real time, either autonomously or explicitly controlled by an online User.

In oder to achieve this goal, we designed an algorithm for exploring efficiently a maze without relying on remembering its path in the maze. There is a lot of applications for this kind of algorithms, for example a cleaner robot in a large area (for now, the algorithm used by the cleaner robot are not really good for too big areas).

We also created a representation for the map that is easy to generate, that could be used by the embedded Artificial Intelligence -even if we didn't prove that concept-, and that can easily be represented and manipulated by a remote User. The utility of mapping a building (who is a priori an accesssible area) are various, for example we can easily immagine an application in a video game (The scan of the area could be directly used in the game), or in the field of remote monitoring.

However, a lot of ideas -about the construction of the map representation or the Artificial Intelligence working with the map- were not fully developed, which can lead to future (research) work.

7 Appendix

1. Robotic Mapping: A Survey,Sebastian Thrun February 2002 CMU-CS-02-111 - `http://robots.stanford.edu/papers/thrun.mapping-tr.pdf`
2. Project page for PyRobot - `https://code.google.com/p/pyrobot/`
3. Project page for oisimple C interface - `http://sourceforge.net/projects/oi-simple/`
4. Page discussing libBiscuit - `https://shanetully.com/2013/04/libbiscuit-a-simple-irobot-create-c-api/`
5. Project page for libcreateoi - `https://code.google.com/p/libcreateoi/`
6. Wikipedia article for a maze solving algorithm - `http://en.wikipedia.org/wiki/Maze_solving_algorithm`
7. Download page from the project MicroMouse - `https://code.google.com/p/maze-solver/downloads/detail?name=MicroMouse-r194.jar`
8. The homepage of the project - `http://graal.ens-lyon.fr/kirc/public/`
9. To view the most important sources - `http://graal.ens-lyon.fr/kirc/public/view_source.php`
10. Git repository for the project - `http://git.aliens-lyon.fr/gcordonn/kirc/`
11. Connectivity work package on the Git repository - `http://git.aliens-lyon.fr/gcordonn/kirc/blob/master/src/connectivity/`
12. Project page for Intrael - `https://code.google.com/p/intrael/`
13. Making a battery pack for MS Kinect - `http://channel9.msdn.com/coding4fun/kinect/Now-were-Kinecting-wit`