Biorobotics Laboratory

Intelligent User Interface for Roombots

Semester project

Maksa David (david.maksa@epfl.ch)

June 7, 2013

Supervisors: Stéphane Bonardi, Rico Möckel, Massimo Vespignani

Professor Auke Jan Ijspeert
Abstract

Roombots are modular robots developed at EPFL in the Biorobotics Laboratory. In the past years different user interfaces were designed to interact with them but there is still room for improvement. The aim of this project was to develop a novel approach to control these robots using a gesture based interface. First we used the RGB camera of a Kinect to detect the position of connectors that Roombots use to make displacements. We converted this information to a 3D graphical representation that shows the position of obstacles and connectors on a rectangular platform. Unfortunately due to lack of time and some unexpected problems we could not finish the detection of Roombots modules and therefor the user interface but the evaluation of the connector detection algorithm showed robustness to many parameters such as the size of connectors seen by the Kinect, the position of connectors, or their deformation due to perspective.
Acknowledgement

I gratefully acknowledge my supervisors Stéphane Bonardi, Rico Möckel and Massimo Vespignani for the technical support and for their valuable advices concerning the project and Emmanuel Senft for his contribution to my early work. I also thank Prof. Auke Ijspeert for the opportunity to realize this semester project in the Biorobotics Laboratory.
# Contents

1 Introduction

1.1 Roombots .................................................. 3
1.2 Previous projects ........................................ 3
  1.2.1 Mobile control interface for modular robots by Gilles Gressier .......... 4
  1.2.2 Mobile control interface for modular robots by Jérémy Blatter .......... 4
1.3 Objectives .................................................. 5
1.4 Robot-user interfaces ...................................... 6
  1.4.1 Gestures for touchscreens ................................ 6
  1.4.2 Gestures for free-form systems ............................ 7
  1.4.3 Gesture recognition with the Kinect .......................... 9

2 Grid detection .................................................. 10

2.1 Description of the grid ..................................... 10
2.2 Description of the task ...................................... 10
2.3 Method ...................................................... 11
  2.3.1 Hough circles algorithm ................................ 11
  2.3.2 The basic algorithm ..................................... 13
  2.3.3 Improved basic algorithm ................................ 14
2.4 Evaluation of the circle detection algorithm .............. 15
  2.4.1 First factor: Distance between the grid and the Kinect ................. 16
  2.4.2 Second factor: position and orientation of the grid on the ground .... 16
  2.4.3 Third factor: perspective deformation ........................ 16
  2.4.4 Fourth factor: Number of detection cycles .......................... 19
  2.4.5 Fifth factor: Illumination .................................. 19

3 3D representation of the grid .................................. 21

3.1 Description of the task ..................................... 21
3.2 Open Scene Graph (OSG) .................................... 22
3.3 Method ...................................................... 22

4 Roombots detection .............................................. 23

4.1 Description of the task ..................................... 23
4.2 Method ...................................................... 23

5 User interface .................................................. 25

5.1 3D modeling with the Kinect ................................ 25
5.2 3D representation ........................................... 26
Chapter 1

Introduction

1.1 Roombots

Roombots are modular robots developed at EPFL in the Biorobotics Laboratory (BioRob). Each robot unit has three degrees of freedom (see Figure 1.1) and has connectors which allow it to connect to an other module’s connector or to a connector on the ground (Fig. 1.2). Roombots can achieve different kind of displacement strategies and they are also able to form a more complex robotic structure which makes them interesting for many research fields such as robot locomotion, modular robotics and swarm intelligence [1]. The longterm vision of the Roombots Project is to use these robots as basic units of adaptive pieces of furniture that can reconfigure themselves [2]. For this reason we need an intuitive user interface to interact with them that most people would enjoy.

![Figure 1.1: Single Roombots unit with 3 degrees of freedom](image1.png)

![Figure 1.2: Roombots modules connect together to form a complex robotic structure](image2.png)

1.2 Previous projects

This semester project focuses on the creation of a new user interface for Roombots. In this section, we review the existing user interfaces for these robots developed at Biorobotics Laboratory (EPFL).
1.2.1 Mobile control interface for modular robots by Gilles Gressier

In this project [3], a graphical user interface based on QtWidgets (Fig. 1.3) was created. It allows to set the position of each degree of freedom of each module separately. It allows also to realize a sequence of rotations and visualization of the position of each module using the Ogre3D engine\(^1\).

**Author:** Gilles Gressier  
**Type of project:** Semester project  
**Year:** 2010-2011  
**Project webpage:** http://biorob.epfl.ch/page-53602.html

![The user interface](http://www.ogre3d.org/)  

Figure 1.3: The user interface\(^2\) shows the position and the orientation of Roombots that the user controls through dials and sliders.

The program created by Gilles Gressier proposes a low level control for Roombots modules and requires the use of a computer. For these reasons it benefits mostly to researchers of the Roombots project. It can be useful for presentations and for simple experiments but the execution of complicated tasks such as the displacement of a single module or several modules from one place to the other would require too much effort from the user.

1.2.2 Mobile control interface for modular robots by Jérémie Blatter

In this project [4], an augmented reality iPad application was developed which allows the user to see the room (in which he is currently staying) through the camera of the iPad and place virtual objects in it such as a virtual table or a chair. Once these objects are placed in the room, the user can see them on the screen of the iPad. Fig. 1.4 shows a snapshot of the application's screen.

This application was created to support a user study on the impact of augmented reality on user experience and the efficiency of the interaction.

**Author:** Jérémie Blatter  
**Type of project:** Master project  
**Year:** 2011-2012  
**Project web page:** http://biorob.epfl.ch/page-75451.html

\(^1\) [http://www.ogre3d.org/](http://www.ogre3d.org/)  
\(^2\) This image was downloaded from the project's webpage.
Although Jérémy Blatter’s application provides a simpler and more intuitive user interface than Gilles Gressier’s program, it’s major drawback comes from the drift of the position measurement of the iPad. Since the position of the virtual object placed in the room is relative to the position of the iPad, all error in the iPad’s position measurement will spread to the virtual object’s position and for this reason after a few seconds that the virtual object was placed in the room it moves away from it’s original position in the room.

Inspired by these different user interfaces we decided to implement a novel approach to control Roombots. We would like to reduce the complexity of the control for the user compared with Gilles Gressier’s program while retaining the liberty of the user to control robots one by one. We also expect the control to be free from drift errors, precise and robust.

1.3 Objectives

The overall goal of the project is to create an interface based on a Kinect tracking of the user, without the need for an external mouse and keyboard based setup. The user will be able to directly select or deselect a group of Roombots modules, to receive a visual feedback from the modules (through LEDs) and to define, on an intelligent grid, destination positions, virtual obstacles or final goal shape. In the scope of this project, the modules will be considered static.

1. Tracking of the user: with a single Kinect, we should be able to retrieve the position and orientation of the different joints of the user. Existing libraries and packages can be used to achieve this goal (see for example [5]). We use libfreenect\(^3\) to communicate with the Kinect.

2. Second Kinect integration: tracking of the user hand gesture and position on the intelligent grid. The two Kinects should be synchronized to use a common coordinate system. An alphabet of interaction will need to be defined, possibly using different modalities (gestures, sounds,...). The notion of ray tracing will be used to find the target position pointed by the user.

3. 3D representation: a virtual representation of the setup need to be created to validate the interaction part (selection, deselection, creation of obstacles,...).

4. Intelligent grid: creation and implementation of a LED grid using several LED drivers and a single PIC controller. The same architecture can be used for the LED attached to the module. The low level communication between the PIC controller and the USB port as well as the low level control of the LEDs by the LED driver are provided.

\(^3\)High level API for the Kinect [https://github.com/OpenKinect/libfreenect](https://github.com/OpenKinect/libfreenect)
Due to lack of time we only completed a part of the original objectives. The majority of the efforts were made on the 3D representation of the setup which can be decomposed as follows:

1. **Grid detection:** detection of the position of the grid of connectors on the Kinect’s RGB image. The algorithm will be able to detect large variety of grids as well as its dimensions.

2. **Roombots module detection:** After the grid detection we place Roombots on the grid. The algorithm should be able to detect their position on the grid.

3. **3D representation:** from the information acquired in the two previous points we reconstruct a 3D model of the setup composed of a grid of connectors, obstacles and Roombots.

### 1.4 Robot-user interfaces

In this section we make a short list of gestures commonly used in user interfaces. The goal of this section is to orient the choice of gestures that we will implement for the Roombots interface\(^4\). These examples were taken from "Designing Gestural Interfaces" by Dan Saffer [6].

#### 1.4.1 Gestures for touchscreens

These gestures are designed for touchpads but we can take inspiration from them to create simple and intuitive robot commands.

- **Tap**
  The tip or pad of the finger touches the surface briefly (<100ms). A double tap performs this gesture twice rapidly, with a (<75ms) pause between the two contacts. It can be used for pushing buttons and selecting.

- **Drag/Slide**
  The tip or pad of the finger moves over the surface without losing contact with the surface. It can be used for drag-and-drop and scrolling.

- **Flick**
  Flick can be done in two ways. In the first way the finger is crooked to start, and then the tip of the finger or part of the finger pad brushes the surface briefly (<75ms) as the finger uncurls. In the second way the finger is straighter and the movement is nearly reversed, with the finger drawing closer to the body and the fingertip or part of the finger pad brushing the surface. Both of these are also called Fling. It can be used to quickly move objects, or to scroll.

- **Nudge**
  The pad of the straight (index) finger slides briefly (<2 seconds) forward. It can be used to move objects.

- **Pinch**
  Two fingers (typically the thumb and index finger) move closer together. It can be used for scaling.

- **Spread**
  Two fingers (typically the thumb and index finger) move further apart. It can be used for scaling.

- **Hold**
  The finger is pressed onto or pointing to an object for an extended period of time. It is also called press and it can be used for selection.

---

\(^4\)This work was made before we restricted our objectives and we decided to present it despite of the fact that we could not take profit from it for the implementation of the 3D representation.
1.4.2 Gestures for free-form systems

These are common gestures and their use does not require any additional object in opposition to touchpad gestures that requires a touchscreen.

- **Nod**
  The head moves up and down in an affirmative movement. This movement does not signify “yes” in certain parts of the word. In the Middle East for example, it signifies the opposite.

- **Shake**
  The head shakes left and right as a negative gesture. It does not signifies “no” in certain parts of the word. In the Middle East for example, it means “yes”.

- **Standing**
  The body is an upright position (Fig. 1.5). It can be used to return to the default mode, or to switch on the program.

- **Sitting**
  It can be used to turn off the program.

- **Hands on hips**
  It can be used to stop an action.

- **Arms up**
  One or both arms are raised straight up above the head. A possible use of this gesture is the increase of a setting or the activation of an object or objects.

- **Arms folded**
  Both arms are folded across the chest. It can be used for stopping an action.

- **Arm(s) in front**
  One arm or both arms are lifted and extended straight forward, parallel to the floor. It can be used to confirm the selection of a choice.

- **Arm(s) out to the side**
  One arm or both arms are lifted to the side, parallel to the floor, creating a 90-degree angle with the torso. It can be used to move a cursor left or right, or to select an object on the left or on the right.

- **Point**
  A single finger is extended outward. Possible uses are the selection or activation of an object.

- **Fist**
  All fingers are curled into a single unit. Possible uses of this gesture are the grasping of an object (drag-and-drop) or the confirmation of an action such as selection or activation.

---

*Image source: http://www.colourbox.com/image/businesswoman-full-body-standing-isolated-on-white-image-3746060*
• **Thumbs up or down**
  While the rest of the fingers are closed in a fist, the thumb protrudes and points up (indicating positive) or down (negative). A thumbs up is offensive in Australia and Nigeria. This gesture can be used to select or to deselect an object.

• **Stop**
  The flat hand is held forward with the palm facing away from the body (Fig. 1.6). (This gesture is very offensive in some parts of the word such as Greece.)

![Figure 1.6: Stop gesture](http://www.123rf.com/photo_16661229_baby-girl-gestures-stop-hand-sign-isolated-on-white.html)

• **Hands folded**
  Both hands slide together, with the fingers of one hand curling and then interlocking with the fingers of the other hand (Fig. 1.7). It can be used to stop an action or to reset the program.

![Figure 1.7: Hands folded gesture](http://www.123rf.com/photo_8701539_close-up-frontal-top-view-of-two-hands-folded-with-intertwined-fingers-praying-on-a-white-desktop.html)

This list is not exhaustive, we could add a lot more examples but before choosing one gesture over another we should first think about what gestures can we detect with the Kinect? This is an important question because if we are not able to implement a robust detection algorithm to segment the fingers of the user than this will completely restrict our choice of gestures. Many articles showed interesting results using different detection algorithms for hand detection and tracking. We present a few examples in Section 1.4.3. Than even if we are able to detect hand expressions we should evaluate the robustness of our detection method for each gesture experimentally.

As we mentioned in Section 1.3 the goal of this project is to implement “ray tracking” to select the object pointed by the user. In addition to this we should implement other gestures to improve the user experience. In my opinion we will probably need to make a trade off between user experience and robustness of the detection because large body movements such as arms up, arms in front or nod are easier to detect but they are less comfortable for the user, while hand expressions (tap, spread, fist...) are more user friendly but also more difficult to detect.

---


1.4.3 Gesture recognition with the Kinect

There has been a lot of research concentrating on hand tracking using the Kinect, as an example we can refer to Tao Hongyong and Yu Youling [7]. They used skin color thresholding and the depth image of the Kinect to segment the hand of the user than they were able to track it with the k-nearest neighbors algorithm. We can also refer to Hui Li, Lei Yang, Xiaoyu Wu, Shengmiao Xu and Youwen Wang [8]. They achieved the recognition of static hand gestures with the depth image of the Kinect and histogram oriented gradient algorithm.

As we finished the introduction part of this report, we can now focus on the presentation of the major achievements made under this project.
Chapter 2

Grid detection

In this semester project we assume that the Roombots modules are placed on a grid of connectors. The detection of this grid’s position, dimensions and orientation is an essential step for the realization of a 3D representation of the setup.

2.1 Description of the grid

The displacement of a single Roombots unit requires a special terrain. This terrain should have connectors specially designed for Roombots to which the Roombots unit can connect to. The displacement of the Roombots unit is easier to achieve when these connectors are aligned and equidistant. The grid of connectors (Fig. 2.1) is specially designed for this reason. The grid shown on Fig. 2.1 is an example. For the grid-detection algorithm that follows, we tried to create a general algorithm that works with grids that have different sizes, orientations, forms or dimensions.

2.2 Description of the task

The first goal was to create a detection algorithm able to detect the position of all connectors of the grid that aren’t hidden by an obstacle or a Roombots module. We used a Kinect device to take images from above the grid (from a fixed position) and the task was to find the position of connectors on the image.
2.3 Method

At the beginning of the development, we took a piece of code written by Emmanuel Senft that demonstrated how to get images from the Kinect and how to display them. The code makes the segmentation of Roombots modules on the depth image and shows the segmented image on the screen. To simplify the detection process, we decided to remove all Roombots modules from the grid because they could hide not only the connector below them but also in some cases the connector next to them (see Fig. 2.2). We could have also use more than one Kinect but it would have complicated much more the detection process.

Figure 2.2: Front view of the setup showing the field of view of the Kinect, some connectors may not be seen by the Kinect.

For this reason we focused our attention on the detection of a simple grid without any obstacles or Roombots on it. For the detection of connectors on the grid, we tried first to use the depth image of the Kinect. If the Kinect is less than 1.2[m] from the grid we can see a different depth in the center of the connectors. Unfortunately when the Kinect is further than 1.2[m] from the grid the quantization step of the depth camera becomes too large to detect the connectors. Consequently instead of using the depth image we decided to use the RGB image of the Kinect for the connector detection.

2.3.1 Hough circles algorithm

A connector on the RGB image looks like a small circle with carvings around it. We decided to use the Hough Circles algorithm to detect the circles of the connectors because it is included in an OpenCV\(^1\) library, that means that we gain time and effort by using it, but this also means that we have less possibilities to adapt the algorithm for our task because we can only change a few parameters in it. (Another solution that we could have used is to put markers on the connectors and than use color blob detection. We preferred to avoid putting markers on the grid.) Before using Hough Circles it is advisable to do some smoothing to improve the robustness of the detection as indicated in the example code for Hough Circles on the dedicated web page\(^2\). We use two different type of smoothing described as follows:

- **Time domain smoothing**
  The Kinect takes 30 images per second, the simplest way to create a very smooth image from these images is to use an exponential low-pass filter.

\[
\begin{align*}
  f_0 &= \text{blackpicture} \\
  f_{t+1} &= 0.9 \ast f_t + 0.1 \ast g_t
\end{align*}
\]

(2.1)

where \( f_{t+1} \) is the smoothed image at time \( t+1 \), \( g_t \) is the image taken at time \( t \) and “+” and “*” are pixelwise addition and multiplication.

\(^1\)http://opencv.org/
\(^2\)http://docs.opencv.org/doc/tutorials/imgproc/imgtrans/hough_circle/hough_circle.html
• **Image-space domain smoothing**
  A Gaussian filter is used for image-space domain smoothing.

After the smoothing, the Hough Circles algorithm is applied to the image. This algorithm is implemented in the OpenCV library. Our task is to call it properly with well chosen parameters. The parameters that we have to choose are the following:

• **The higher threshold for the internal canny edge detector**\(^3\) (The lower threshold is just the half of the higher threshold). After some experimentation this parameter was set to 40.

• **The minimum distance between detected circles**
  In this stage of the processing we have no information about the distance between two connectors because the size of the grid can be bigger or smaller on the image depending on how far the Kinect is from it. For this reason, this parameter is set to zero, later we use a condition to filter out circles that are too close to each other, see in Section 2.3.3.

• **The maximum circle radius**
  In this stage of the processing we can only assume that the radius of connector circles on the image is smaller than 7 pixels. This assumption was tested and it is correct even when the Kinect is only 70cm from the grid. For this reason it makes sense to put a higher limit on the size of detected circles. Furthermore if there’s no limit, the Hough Circles algorithm will take a lot more time to compute because it will try to evaluate circles that has bigger size.

• **The minimum size of circle radius**
  The minimum size of circles radius parameter is set to zero. In Section 2.3.2 we will use a condition that eliminates circles that are too small to be connectors. Consequently for now there is no need to set this value higher than zero.

We observe that circle detection using Hough circles algorithm on the RGB image gives very different results each time we run the algorithm: sometimes a lot of circles are detected and other times there is only one or two detected circles returned by the algorithm. A few examples are shown on Figures 2.3a and 2.3b (Detected circles are drawn on the image which is the source image for the circle detection.)

![Figure 2.3](image-url)

Figure 2.3: A different set of circles was detected by the Hough circles algorithm for almost the same source images

\(^3\)Optimal algorithm to detect high contrast regions (edges) on an image (http://en.wikipedia.org/wiki/Canny_edge_detector).
This is very surprising since even if the images look exactly the same Hough Circles algorithm can return very different results for each of them. Probably this is due to some noise while recording or transferring the image, because for the same image Hough Circles algorithm returns exactly the same circles. Fortunately we also observed that not always the same circles are returned again and again but after a few hundred of executions of the Hough Circles algorithm (on different images taken at different moments) all circles were detected at least once. This observation inspired the following algorithm.

2.3.2 The basic algorithm

The Kinect takes an RGB image and after smoothing it we run the Hough Circles algorithm on the image. It returns the position of the center and the radius of each circle. All these circles’s radius is smaller than seven pixels as stated before in Section 2.3.1. We assume that the largest of them is a connector circle. Then we can eliminate circles that are too small compared to this largest circle. In many cases, Hough Circles algorithm detects screws and carvings as circles even if they obviously does not look like a circle. Since they are usually smaller than connector circles this algorithm will be able to eliminate them. The major drawback of this method is that it works only if the Hough Circles algorithm returns at least one connector circle.

Once we eliminated these falsely detected circles we are going to save in memory the location of the rest of them. A simple way to do this is to draw them on an image that has the same size than the RGB image that comes from the Kinect. For this purpose we have chosen a grey image and the location of detected circles are drawn as small white dots (2 pixels radian) on this image. The choice of this method makes sense because the next step of the processing is to repeat all operations from the beginning: the Kinect takes an other RGB image from the same location, after smoothing, Hough Circles algorithm is used, we eliminate circles that are too small compared to the largest detected circle and finally draw their location on the same grey picture that stores the results from the previous cycle. If the algorithm detects circles that are already on the grey image, it will simply draw a white dot on top of the previous white dot and there is no need to use conditions to eliminate one of them. The other advantage of this method is that when the process is running we can see the results in real time on the grey image. We can see which circles are detected, where the noise comes from and this gives the ability to debug and set the right parameters for this method more easily.

Figures 2.5a and 2.5b present a few results from different detections after a few hundred cycles.
2.3.3 Improved basic algorithm

At this stage of the processing the algorithm detects a lot of circles that are not connector circles but we can improve it using the condition that connector circles are aligned and equidistant.

To make this method computationally efficient, the idea is to take the location coordinates of detected circles just before they are drawn on the grey image (see Fig. 2.7) because afterwards we won't keep this information and we would be obligated to search them on the image. Then for all circle center point P1 and P2, the algorithm computes the vector $v = P1 - P2$, it checks if the norm of $v$ is larger than 20 pixels $^4$ and computes the coordinates of the point $C = P1 + v$. Finally if the grey image contains a white pixel (circle location$^5$ from a previous detection) at the location $C$ then we can assume that $P1$, $P2$ and $C$ are aligned and equidistant detected circles (example on Fig. 2.6) and therefore we mark them with a large grey dot (see examples on Figures 2.8a and 2.8b).

A drawback of this method is that it supposes that each connector is part of a group of three connectors that are aligned and equidistant.

$^4$This parameter was determined experimentally

$^5$The circle location is not a one pixel area but a two pixel radius white point. This radius was determined experimentally.
2.4 Evaluation of the circle detection algorithm

This section intends to describe the performance of the detection algorithm described in the previous sections. We tested the algorithm in different conditions in order to acquire qualitative knowledge of its limits and its capacity of detection. We were mainly interested in the factors that affect the most the performance of the algorithm and we present our results in the following lines.
2.4.1 First factor: Distance between the grid and the Kinect

First we were interested in how the distance between the Kinect and the grid affects the detection. This factor will change the size of the grid on the RGB image. It will not only change the size of the connector circles but also the distance between them.

We expect that the detection will be successful between a higher and a lower distance limitations. Our experiences are focusing on the determination of these limits and the verification of the pertinence of the evaluation method.

Fig. 2.9 shows the results of our experiments. We made ten experiments for each distance (50cm, 55cm, 60cm, 70cm, 80cm, 100cm, 120cm, 140cm, 160cm) and counted the number of successful detections out of ten. The detection was counted successful when all 16 connectors of the grid were detected and no error occurred. There are two types of error that we observed: in some cases the algorithm could not detect all circles even after two minutes of continuous detection and in other cases the algorithm made false detections. As shown on Fig. 2.9 for 60cm the algorithm made a better score than for 70cm but only by one detection which is not significant. At 55cm the performance of the algorithm drops as the radius of connector circles becomes bigger than seven pixels that was a threshold for the Hough Circles algorithm (Fig. 2.10). We did not make additional measures to find highest distance limit where the performance of the detection algorithm would drop because our experimental setup did not allow to fix the Kinect higher than 160cm from the ground.

2.4.2 Second factor: position and orientation of the grid on the ground

Since we haven’t made any assumption regarding the position of connector circles on the image in the detection algorithm, the position and the orientation of the grid should not affect the detection as all connector circles are inside the field of view of the Kinect. This hypothesis was confirmed after ten experiments in which we put the grid in each corner of the image an rotated it three times by 45° and three times by 22°.

2.4.3 Third factor: perspective deformation

This factor changes the position of the connectors on the RGB image in a way that connectors that are aligned and equidistant on the grid may no longer appear aligned and equidistant on the image. To evaluate the perspective deformation, we can apply the thin lens law \[9\] to the Kinect’s camera that corresponds to Equ. 2.2.
Figure 2.10: Noise due to the texture of the grid

![Image of Figure 2.10]

Figure 2.11: Thin lens law [9]

![Image of Figure 2.11]

Figure 2.12: View from the side of the setup. The grid is tilted by an angle phi.

![Image of Figure 2.12]

\[ M = \frac{f}{f - d_o} = \frac{d_i}{d_o} = \frac{h_i}{h_o} \]  

(2.2)
Values \( f, d_o, d_i, h_o \) and \( h_i \) are represented on Fig. 2.11. From relation 2.2 assuming that the focal length \( f \) is very small compared to the distance between the connector and the kinect, we can obtain:

\[
h_i = \frac{f}{f - d_o} \times h_o \approx \frac{f}{d_o} \times h_o
\]  

(2.3)

For connectors \( C_0, C_1 \) and \( C_2 \) (represented on Fig. 2.12), \( h_i \) becomes respectively \( h_{i,0}, h_{i,1} \) and \( h_{i,2} \):

\[
h_{i,0} = 0
\]  

(2.4)

\[
h_{i,1} = f \times \frac{a \cos \phi}{d_o + a \sin \phi}
\]  

(2.5)

\[
h_{i,1} = f \times \frac{2a \cos \phi}{d_o + 2a \sin \phi}
\]  

(2.6)

Fig. 2.13 shows \( h_{i,1} \) and \( h_{i,2} \) depending on \( \phi \) (Fig. 2.12). These values were computed by taking:

- \( a = 11 \text{ cm} \)
- \( f = \frac{\text{focal length}}{\text{pixel size}} = 518 \) (Focal length and pixel size were taken from [10])
- \( d_o = 140 \text{ cm} \)

![Figure 2.13: Position on the image \( h_i \) of connectors \( C_1 \) and \( C_2 \) depending on \( \phi \)](image)

If the distance on the image between the connector \( C_0 \) and \( C_1 \) is more than two pixels smaller than the distance between connectors \( C_1 \) and \( C_2 \) the detection algorithm will no longer consider \( C_0, C_1 \) and \( C_2 \) equidistant. Consequently it won’t be able to make the detection of these connectors. Fig. 2.14 shows the deformation error for different \( \phi \) and Kinect-grid distances. The deformation error is the difference between the distance (on the image) between connectors \( C_0 \) and \( C_1 \) and the distance between connectors \( C_1 \) and \( C_2 \).

The critical limit (\( \pm 2 \) pixels) is represented by two red lines.

As shown in Fig. 2.14 the perspective deformation can be reduced effectively by increasing the distance between the Kinect and the grid. Due to lack of time we only made experiences for one particular Kinect-grid distance that was 140cm. We made five experiments for \( \phi = 23^\circ \) and an other five for \( \phi = 35^\circ \). For \( \phi = 23^\circ \) the detection worked perfectly each time probably because we used a four by four grid and the perspective distortion only affected one direction. However for \( \phi = 35^\circ \) the algorithm made at least one error each time.

---

6Because the experimental setup permitted to reproduce these angles and later we observed that these measurements are relevant for the evaluation of the detection.
These experiments are consistent with the theoretical model that we developed before but there are some other factors that we did not take into account for the estimation of the perspective distortion, such as the position of the grid on the ground or its orientation but a more complex analysis is out of the scope of this project.

2.4.4 Fourth factor: Number of detection cycles

The Kinect repeats the same detection operations in a cycle as described in Section 2.3.2. After each cycle the detection improves but it takes time since the Kinect captures 30 images per second and therefore we can only make 30 cycles per second. Fig. 2.15 represents the number of detected circles out of 16 depending on the duration of the detection. We observe that most circles are detected after 8 seconds than there is no further detection for approximately 50 seconds and finally at one minute we have the detection of the last few remaining circles. The reason why we detect the last circles after 60 seconds is that in the experiment we shake gently the setup at 60 seconds which shifts the Kinect’s image and allows a better detection.

2.4.5 Fifth factor: Illumination

We repeated the previous experiments of Section 2.4.4 with a different illumination condition in which the image is less illuminated and we do not observe the same contrast between the connector circles and the...
grid. In this condition the performance of the algorithm decreased as expected. Fig. 2.17 represents the result of the experiments carried out with the same method as in section 2.4.4. In my opinion this decrease of performance is mainly due to the loss of contrast between the circles and the grid and the presence of shadow on the grid.

![Figure 2.16: Source image for the detection in a different illumination condition.](image)

![Figure 2.17: Number of detected connectors out of 16 depending on the duration of the detection with a less contrasted source image (Figure 2.16)](chart)

As we presented in this evaluation, many factors can influence the performance of the detection and therefore in many scenarios it is difficult to predict the final result of the processing but in my opinion for the purpose of this project this algorithm is sufficiently accurate.
Chapter 3

3D representation of the grid

In the second chapter of this report we discussed how the algorithm detects connectors on the image and draws their location on the grey image. As we described, it is a continuous task. The algorithm repeats the same cycle detecting more and more connectors and it will stop when the user pushes the create 3D button. That means that the user should watch the evolution of the process and when all connector circles are detected the user should stop the process\(^1\). Depending on light conditions in some cases the algorithm detects all circles in less than 5 seconds while in other cases it takes more than one minute therefore it is better to let the user stop the program instead of using a fixed runtime limit.

In this chapter we are going to describe what happens when the user pushes the Create 3D button.

3.1 Description of the task

On the grey image, the algorithm marked the location of all detected connectors with a large grey dot (Fig. 3.1a). Our task is to convert this information to a 3D graphical representation of the grid (Fig. 3.1b).

Figure 3.1: The image that we transform to a 3D representation (Fig. 3.1a) and the result we should obtain for this particular image (Fig. 3.1b)

\(^1\) Chap. 5 describes the user interface with more details
3.2 Open Scene Graph (OSG)

For the 3D representation we use OSG\textsuperscript{2} which is an open source 3D graphics toolkit written in C++. Under this project we developed a program that creates the 3D representation from data written in a textfile. The textfile contains the dimensions of the grid, the discrete position corresponding to obstacles and the position and orientation of detected Roombots (Chap. 4).

3.3 Method

As mentioned in Chap. 2, we assume that connectors are aligned and the distance between two neighboring connector is always the same. First we compute the smallest vector between detected connectors $v_1$. To make this computationally efficient, this vector is calculated in the algorithm presented in Section 2.3.3. Then we create the vector $v_2$ perpendicular to $v_1$ and having the same norm as $v_1$. Finally we detect the position of one large grey dot on the grey image that we set as the origin and we check for any linear combination of $v_1$ and $v_2$ if there is a connector on the grey image.

Thanks to this method we can convert the grey image space coordinates of the connector to a discrete space that contains all information we need to reconstruct the 3D model of the grid. For simplification of the 3D representation we assumed that the form of the grid is the smallest rectangle that contains all connectors and all other positions on the grid where we did not detect connectors will appear as an obstacle (Fig. 3.2).

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{figure3.png}
\caption{Two obstacles represented by black cubes in the 3D environment.}
\end{figure}

For the implementation of this method two main difficulties were encountered: first the vectors $v_1$ and $v_2$ are not exactly aligned to the position of the connectors. They usually contain some error and when the algorithm computes a linear combination of these vectors it will add the errors and create a drift. Consequently large grey dots that are too far from the origin won’t be detected. The other problem is the image deformation due to the Kinect’s camera and the perspective deformation.

The simplest solution to these problems is to use the largest possible grey dots to represent the location of the connectors and that each time the algorithm detects a large grey dot, we correct the error of the drift.

The result of the detection is written to a textfile by the program. The 3D representation is generated by another program that we presented in Section 3.3. The reason why we use a separated program for the 3D representation of the setup is that we wanted to simplify as much as possible the programming part of this project. We decided that the integration of the OSG graphical window to the existing user interface can be achieved later once the user interface is finished. Unfortunately another task took a lot more time than we expected which did not allow us to integrate it. This task is the Roombots detection and we present it in the next chapter.

\footnote{\url{http://www.openscenegraph.org/}}
Chapter 4

Roombots detection

In this chapter, we present the approach we adopted to detect the position of Roombots units on the grid using the Kinect.

4.1 Description of the task

As we mentioned in Section 2.3, before detecting Roombots modules, we detect the position of all connectors of the grid. Afterwards we place one Roombots unit on it and we want the program to detect to which connector(s) the Roombots module is connected to. We can repeat this task several times and each time add one module to the grid so that we can detect as many robots we need.

4.2 Method

For the implementation of this algorithm we use a code written by Emmanuel Senft that is able to make the segmentation of Roombots or any other objects using the depth image of the Kinect. His method takes an image before we put the robot on the grid and takes an other image with the robot. Than his algorithm compares the two images and from the difference it is able to make the segmentation of the additional object. Fig. 4.1 shows an example of the segmentation of a book.

Figure 4.1: Segmentation of a book
The segmentation becomes more complicated when we put more than one new object on the image because they could overlap and it would be difficult to separate them. For this reason we preferred to restrict the use of this detection to one particular robot at a time.

We combine the segmented image with the position of connectors that we obtained during the detection of the grid to see which connectors are under the Roombots module.

Despite the simplicity of this approach we observed that the algorithm was not able to detect any Roombots modules correctly. In fact the source of the problem resides in the shift of the RGB image and the depth image of the Kinect. We used the RGB image for the detection of the grid therefor the location of connectors is expressed in the referential of the RGB image, while for the segmentation of the Roombots, we use the depth image. We measured the shift between these two images and realized that the shift is between 0 to 30 pixels depending on the area of the image.

We first searched on the Internet and in the documentation of libfreenect\(^1\) for the explanation of this phenomenon knowing that many programs use the superposition of the RGB and depth images with no apparent difficulty. Finally by comparing with an example code provided with libfreenect, we managed to find the solution for this problem. In fact we had to change the format of the depth image to: “FREENECT\_DEPTH\_REGISTERED” which aligned the depth image to the RGB image.

Although by changing the format of the depth image it’s pixels align to the pixels of the RGB image, this modification also changes the depth resolution. Fig. 4.2 shows the relation between the pixel value on the depth image and the distance of the corresponding object to the Kinect after the modification of the depth format.

![Figure 4.2: Depth image pixel value depending on the distance from the Kinect](image)

As shown on Fig. 4.2 the resolution of the depth image becomes insufficient to detect Roombots if they are further from the Kinect than 110cm. This problem appears only if we set the format of the depth image to “FREENECT\_DEPTH\_REGISTERED”. In the example code provided with libfreenect the resolution of the depth image is sufficient to detect Roombots even 160cm from the Kinect, which means that this issue does not come from hardware limitation but there is probably a parameter to change in the settings of the depth camera. For this reason I have also tried different depth formats and searched more in the documentation but I did not find solution to this problem. Due to lack of time I gave up the continuation of this algorithm.

---

Chapter 5

User interface

The goal of this chapter is to present the user interface of the program developed under this semester project. The code is written in two separate QtProjects: one of them handles the detection, the other the 3D representation using OSG. The program that handles the detection creates a textfile called state.txt in the data folder that contains the size of the grid and the position of obstacles on the grid. The other program reads this file and creates the corresponding 3D representation.

5.1 3D modeling with the Kinect

This program was developed to detect the grid of connectors. It’s user interface is presented on Fig. 5.1.

![Figure 5.1: Presentation of the user interface](image)

When the program launches, it begins to detect automatically the position of connectors on the RGB image. The user interface shows the RGB image seen by the Kinect on the lower left corner. On the top left corner it shows all the connectors detected from the beginning of the program (as large grey dots) and on the right it shows circles (in blue) detected in the current detection cycle by the Hough Circles algorithm. This representation gives the possibility to the user to visualize the results of the detection algorithm in real time.

Once all connectors are detected, that takes normally less than 15 seconds but it depends a lot on light conditions (see Section 2.4.4 and Section 2.4.4), the user should stop the detection process by pushing the "Create 3D" button. This action will not only stop the process but it will also count the number of detected
connectors on the grey image (represented in the upper left corner of the user interface) and write in the text file (i.e. “state.txt”) the size of the grid and the position of the obstacles. The program completed it’s task properly if all large grey dots are marked with a white circle as represented on Fig. 5.2.

![Image of the user interface with marked grey dots](image1.png)

Figure 5.2: When the user pushes the “Create 3D” button all grey dots are marked with a white circle.

The user interface also includes a “Pause” button to stop the process. A second click on the “Pause” button will continue the detection. There is also an “Erase Connectors” button to erase the content of the grey image an therefor it is equivalent to restart the process from the beginning. Finally to quit the program we use a “Quit” button.

### 5.2 3D representation

This program uses OSG presented in Section 3.2 to generate a 3D model from a text file. The text file contains the dimensions of the grid, the position of obstacles and the position of detected Roombots. Although the detection of Roombots is not working, we can write their position manually in the text file and create different 3D representations. Fig. 5.3 shows an example.

![Image of a 3D representation](image2.png)

Figure 5.3: Example of 3D representation realized by the program.

26
Conclusion

The initial goal of this project was to create a novel user interface for Roombots however due to lack of time and some unexpected problems we had to restrict our objectives. We decided to focus on the improvement of the grid detection algorithm that showed robustness to many different parameters in the final evaluation. Indeed the evaluation of the detection method showed that in optimal light conditions nearly all connectors are detected after 15 seconds. The detection is independent to the position of the grid on the image and it is also independent to the orientation of the grid as the deformation error due to perspective is less than 2 pixels on the image.

We think that the program developed under this project will provide a robust basis for the implementation of a gestural user interface for Roombots. In fact the selection of any object using the “ray tracking” approach (i.e. the user selects an object by pointing to it) can only be achieved if the program has a fine knowledge about the position of this object. For this reason I think it was necessary to improve the performance and the robustness of the detection algorithm.
Bibliography


