Semester Project

Study of new Roombots modules

Biologically Inspired Robotics Group

Simon Blanchoud
Supervisor : Prof. Auke Jan Ijspeert
Assistant : Dr. Masoud Asadpour

February 18, 2007

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 The Roombots Project

The Roombots project is a very vast project: its goal is to create self-reconfigurable furniture using modular robotic. The main idea of the modular robotic is to have many modules that cooperate together in order to create a complex global behavior, or a structure in our case. This project is a collaboration between Microsoft Research Cambridge[1] and the Biologically Inspired Robotics Group[2] (BIRG).

1.2 Goals Of The Project

This semester project was part of the Roombots project. The main goal of this project was to develop a new module that could be used for reconfiguration in the Roombots project. This project focused on the suitable capabilities of the modules and not on their hardware implementation; it was developped entirely in simulation.

In order to find the most suitable module for our task, I had to :

1. Create the simulated model of each module
2. Write the program to control each module
3. Determine the desired behaviors and the way to test them

[1] http://research.microsoft.com/cambridge/
[2] http://birg.epfl.ch/

4. Rank the modules according to these tests

Finally, I analyzed more deeply the movements of the more promising module.

1.3 Tools

This project was developped using the following tools :

Webots [3] A powerfull simulation software which includes a modelling tool, ODE[4] and various devices that allowed me to model the modules

C language All the codes I wrote for this project used this language

Pen and paper Used to study possible movements and combinations of movements of the modules. I used this when it was faster than modeling it in Webots

Paper cube Made using the origami technique [5], I used them when the "Pen and paper" tool was not enough

LaTeX [6] Used to write this report

Gimp, Xfig, Inkspace [7] Used to create the pictures and the graphs of this report

1.4 State Of The Art

In the field of modular robotic, the latest developments on self-reconfigurable robots are represented by the following modules :

M-TRAN III [8] In this project, carried out by the Distributed System Design Research Group from the National Institute of Advanced Industrial Science and Technology (AIST) of Japan, the modules use permanent magnets as connection mechanism. They have two axes of rotation and six connection surfaces (see figure 1.1)

[3] http://www.cyberbotics.com/
[4] The Open Source physics engine used by Webots, http://www.ode.org/
[5] http://www.mathematische-basteleien.de/oricube.htm
[6] http://www.latex-project.org/
[7] http://www.gimp.org/, http://www.xfig.org/, http://inkscape.org/
[8] http://unit.aist.go.jp/is/dsysd/mtran/English/

CONRO [9] In this project, carried out by the Polymorphic Robotics Laboratory from the University of Southern California, the modules use a passive connector, which has pins, and an active connector that contains the connection/disconnection mechanism. They have one rotation axis for each connector and the active connector can attach to three faces on the passive connector (see figure 1.2)

PolyBot G3 [10] In this project, carried out by the Palo Alto Research Center (PARC) in California, the modules use grooved pins and holes as connection mechanism. Each module has two connecting faces and one axis of rotation (see figure 1.3)

None of these modules have been used to explore the possibilities of adaptative furniture.

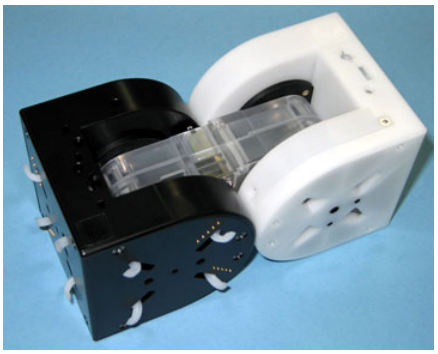

Figure 1.1: A M-TRAN III module

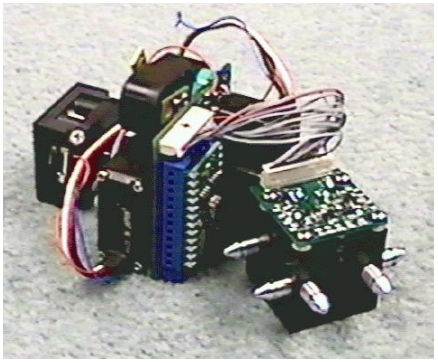

Figure 1.2: A CONRO module

[9] http://www.isi.edu/robots/conro/
[10] http://www2.parc.com/spl/projects/modrobots/polybot/g3.html

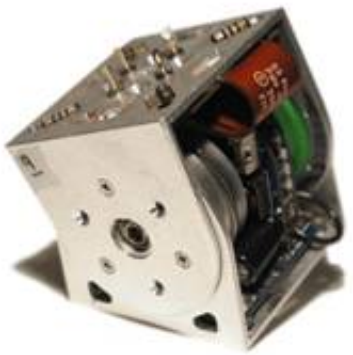

Figure 1.3: A PolyBot G3 module

Chapter 2

The Modules

In my study of new modules I started with five different models; all of them, except Cube4, were proposed by Mr. Asadpour. They all have cubic shapes considering that cubes facilitates the self-reconfiguration process :

- The regular structure of the cube allows the modules to create a dense and compact structure
- As the distance between two modules of the structure is the size of the travelling module, it allows them to stay aligned and to connect easily
- Having all the modules with the same shape allows them to get mixed. This has not been studied in my project

Remarks :

1. The faces of all the modules have been numbered according to the convention explained by figure 2.1
2. In all the schematics representing the modules, the red parts represent the moving parts of the model and the red arrows represent their corresponding axis of rotation (or translation in the case of Cube4)
3. In the simulation the modules use Hermaphrodite Connectors [1] in order to attach to each other

[1] A generic genderless connecting device, named after the corresponding device in Webots

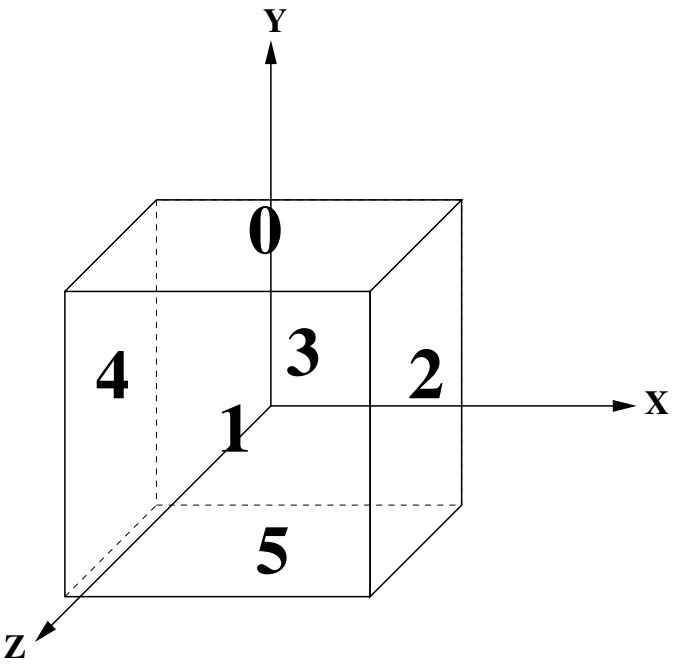

Figure 2.1: Numbering of the faces for all modules

2.1 Cube1

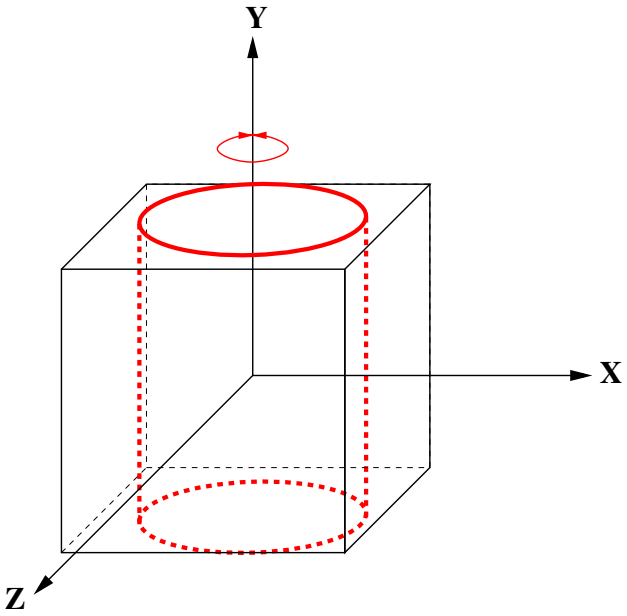

Figure 2.2: The Cube1's schematic

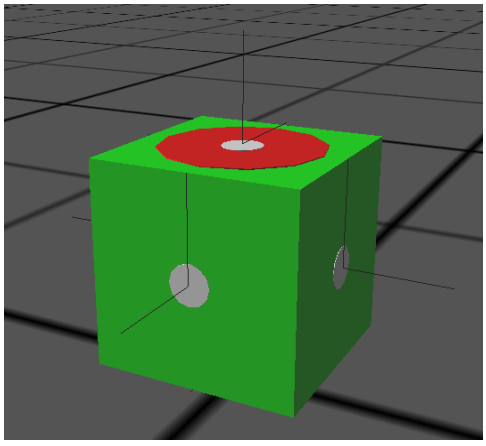

Figure 2.3: The Cube1 in simulation

Its principal characteristics are :

- One degree of freedom
- Two rotating faces, which are in fact only one rotational servo
- The servo has no angular limitations

- One Connector per face

This module is the simplest one you can think of but this simplicity is what is interesting about it. By having such a simple robot we can produce very cheap real ones. This would allow us to compensate the possible drawbacks by the number of modules.

2.2 Cube2

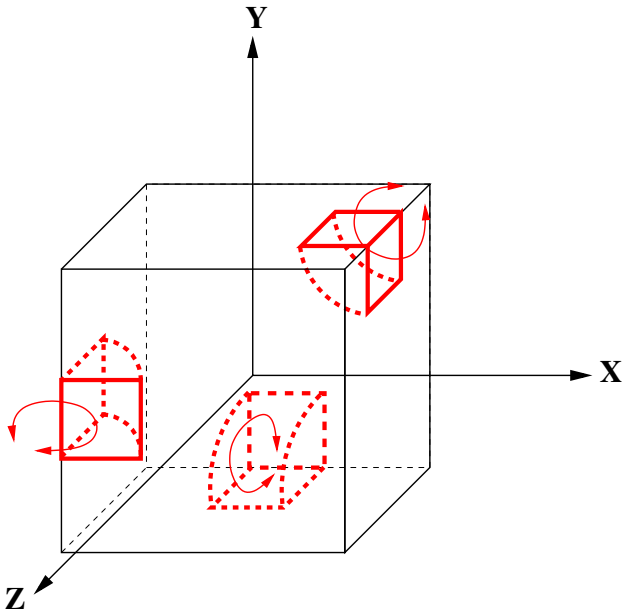

Figure 2.4: The Cube2's schematic

Its principal characteristics are :

- Three degrees of freedom
- Six rotating faces, which are in fact only three rotational servos
- The servos can rotate between -180 and +180 degrees
- Four Connectors per face

This module is the most complex one and, as it has four times more Connectors, probably one of the most expensive to produce too. But by having its axes of rotation on the edges of the cube (see figure 2.4) this allows it to perform movements that cannot be done by any other module. It is also the only module which can move on all its faces.

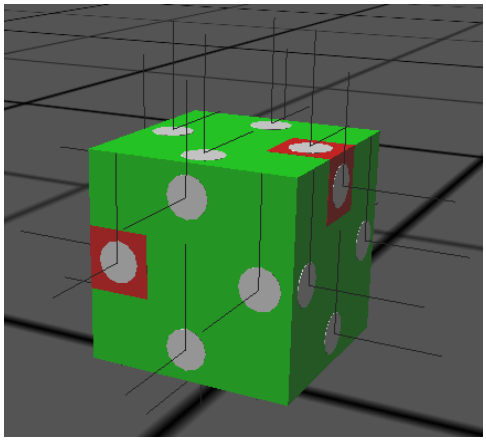

Figure 2.5: The Cube2 in simulation

2.3 Cube3

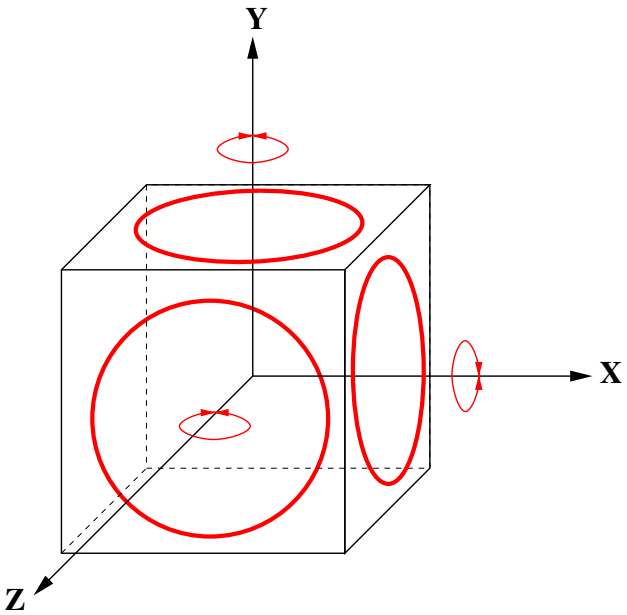

Figure 2.6: The Cube3's schematic

Its principal characteristics are :

- Three degrees of freedom
- Three rotating faces, which are three rotational servos
- The servos have no angular limitations

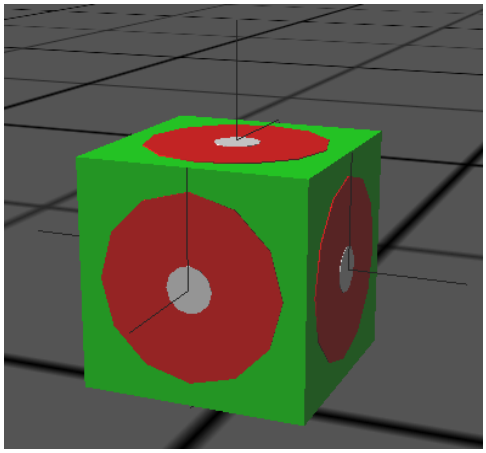

Figure 2.7: The Cube3 in simulation

- One Connector per face

This module is an extention of Cube1, the main idea was to see if by increasing the number of degrees of freedom we would also increase the capabilities of the module.

2.4 Cube4

Its principal characteristics are :

- Three degrees of freedom
- Three moving faces, which are three linear servos
- The servos can move up to one time the size of the cube
- One Connector per face

The purpose of this module was to explore the capabilities of a module that uses linear servos instead of rotational ones. I made some tests with it but it became quickly clear that it was not adapted to self-reconfiguration : The main problem was that the structure, which is made of modules, is unable to help the modules that are moving on it. When moving without the structure, we need two modules for each direction in which we want to be able to move; so if we want to be able to do everything with a group of modules, this group must be quite big. Moreover, the group of modules

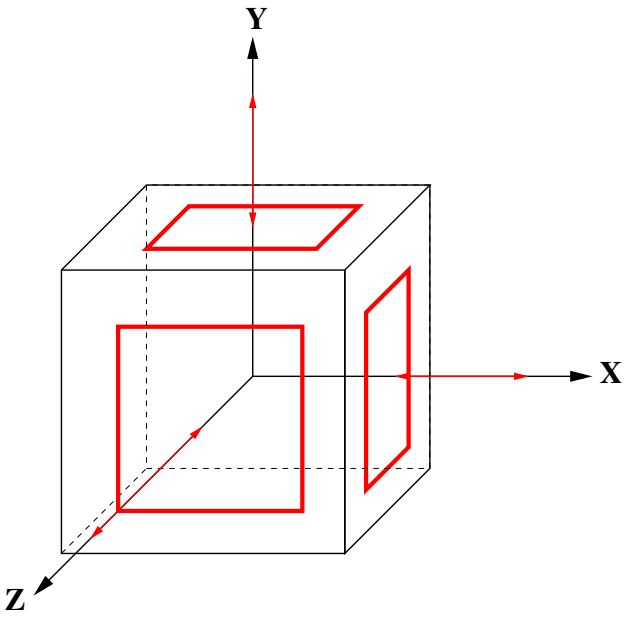

Figure 2.8: The Cube4's schematic

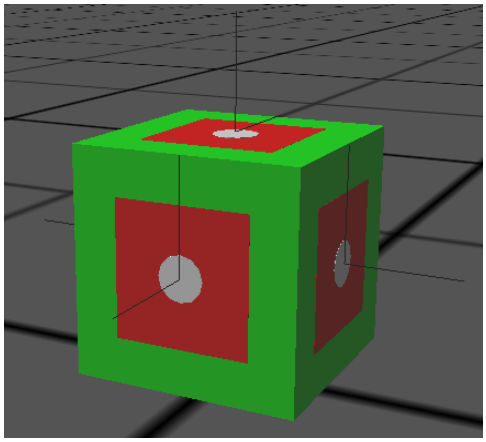

Figure 2.9: The Cube4 in simulation

should be able to move over the structure in order to reach a suitable position. But as soon as the moving group will drop one of its modules in its place, it will also lose one of its moving direction.

This module is not included in the results of this study as I dropped it before I made all the tests.

2.5 Cube5

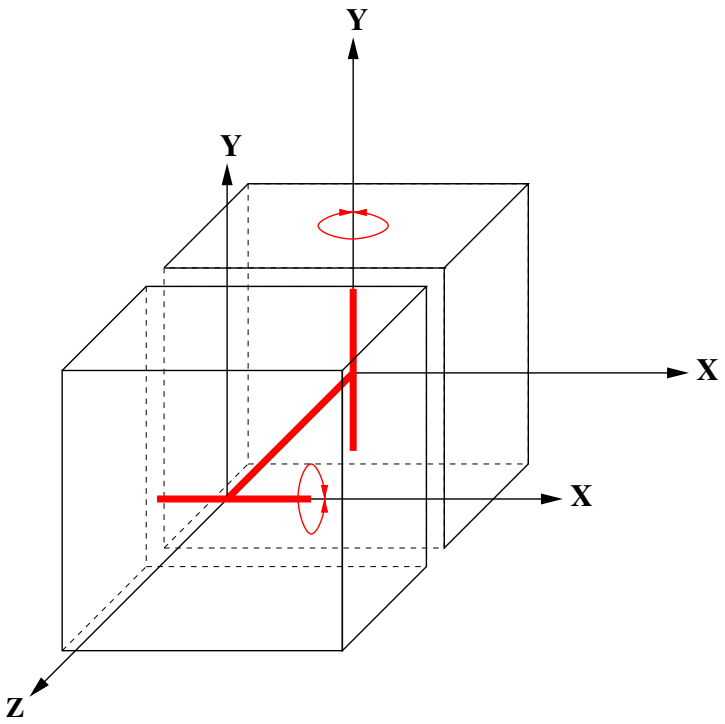

Figure 2.10: The Cube5's schematic

Its principal characteristics are :

- Two degrees of freedom
- The two cubes can rotate around each other using two rotational servos
- The servos have no angular limitations
- Four Connectors on the side faces and two on the other ones

The idea of this module is that it would be able to move on its own even without the help of the structure. I did not used it in my tests because you

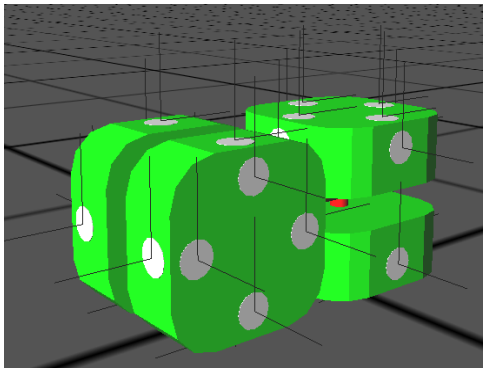

Figure 2.11: The Cube5 in simulation

need to leave some space between the two cubes of the module for them to be able to rotate. The problem is that, because of this small space, the total length of the module is not anymore a multiple of the size of a cube. This leads to great connection problems when moving on the structure (see figure 2.12).

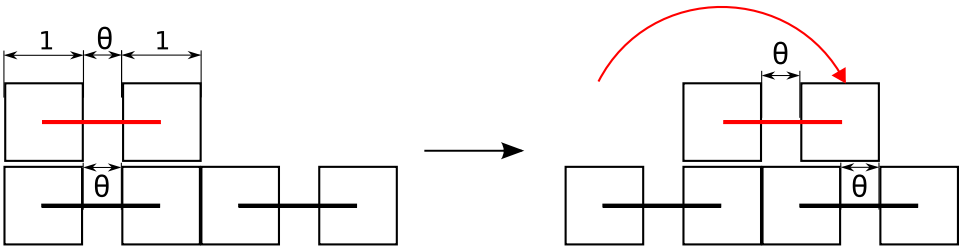

Figure 2.12: The Cube5 moving over the structure

You can reduce this distance by rounding the angles of the cube. The problem is that if you want to reduce it to zero the cube will become a cylinder. Moreover if you want both cubes to be able to move freely you will need to transform these cylinders into spheres. It is a lot more difficult to create a resistant structure using spheres than using cubes.

This module is not included in the results of this study as I dropped it without having done any test with it.

Chapter 3

The Controllers

There are many ways to control a reconfiguration process. Of course the ideal method would be to let the modules deduce on their own what they need to do, but this is very difficult to achieve. Actually searching how the modules could find the reconfiguration sequence, and everything related to this, is the biggest part of the Roombots project and it is being explored by Mr. Asadpour. In order to simplify the work for the moment, we decided to develop a simple language for defining the movement sequences. I implemented two different techniques to control the modules.

Remarks :

1. In addition to these controllers I also wrote a small plugin for Webots that filters ODE collisions to speed up the simulations as much as possible
2. As these controllers have been developped to suit the needs I encountered during the tests I made on the modules, only the three models that I have fully explored (Cube1, Cube2 and Cub3) have them implemented

3.1 Decentralized Controllers

This controller is the closest one to the ideal solution. The basic idea is that every module have the same controller. The controller is a finite state machine that, depending on the identification number (ID) of the module, starts in the appropriated state.

```
if (state == DETACHING) {
  detach(5, ONLY);
  rotate(2, -90);
  next_state = ROTATING_LOCK;
} else if (state == ROTATING_LOCK) {
  if (in_place(2) == 1) {
    attach(3, ONLY);
    detach(5, WITHOUT);
    rotate(2, 90);
    next_state = ROTATING;
  }
} else if (state == ROTATING) {
  if (in_place(2) == 1) {
    attach(3, WITHOUT);
    attach(5, WITH);
    next_state = IDLE;
  }
}
```

This code, which is for Cube2 modules, makes the module number 1 move using its lock number 2. The initial state is `DETACHING`.

The interaction with the modules is made using the following interface :

- `int presence(int face_id)`: Returns 1 if there is another face close enough to get connected to the face specified by the argument
- `void attach(int face_id)`: Atttaches the face specified by the argument
- `void detach(int face_id)`: Detaches the face specified by the argument
- `void rotate(int servo_id, float angle)`: Rotates the servo specified by the first argument by the angle, in degrees, specified by the second argument
- `int in_place(int servo_id)`: Returns 1 if the servo specified by the argument has reached its objective position
- `void send_ack(int face_id)`: Sends an acknowledgment bit on the face specified by the argument
- `int receive_ack(int face_id)`: Returns 1 if an acknowledgment bit has been received on the face specified by the argument

Remarks :

1. For Cube2, there is a second argument for the functions `presence`, `attach` and `detach` which is of type `Behaviors`. It can have the values `WITH`, `WITHOUT` and `ONLY` and it defines if the action needs to be performed on the Connectors of the face `WITH` the one of the servo, `WITHOUT` it or `ONLY` for this Connector
2. In order to implement a sequence using this type of controller, you have to define the corresponding finite-state machine. Working on the directed graph that represents the finite-state machine can be easier. The code of the modules contains already everything needed to translate the machine, or the graph, into the code that will make the modules perform the sequence

3.2 Centralized Controllers

In these contollers, a Supervisor[1] has been added. It directly tells every module what to do when the module has to do it (see figure 3.1). Every module have the same controller but it only listens to the commands of the Supervisor. This gives more comprehensive sequences as they are totally linear.

```
detach(1, 5, ONLY);
rotate(1, 2, -90);
has_rotated(1);
attach(1, 3, ONLY);
detach(1, 5, WITHOUT);
rotate(1, 2, 90);
has_rotated(1);
attach(1, 3, WITHOUT);
attach(1, 5, WITH);
```

This code, which is for Cube2 modules, makes the module number 1 move using its lock number 2.

The controllers of the modules are not the interesting part because you never really use them: you always use the functions of the Supervisor in order to control the modules. It interacts with the modules using the following interface :

[1] In Webots this is a program that controls a world and its robots and that has extended capabilities, compared to standard robots

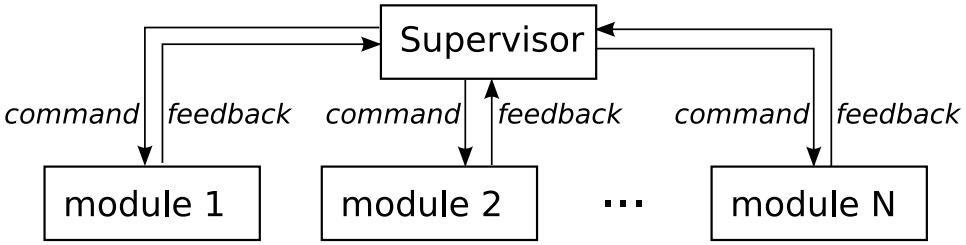

Figure 3.1: The Supervisor - modules interactions

- `void attach(int module_id, int face_id, int behavior)`: Tells the module, whose ID is specified by the first argument, to attach its face specified by the second argument
- `void detach(int module_id, int face_id, int behavior)`: Tells the module, whose ID is specified by the first argument, to detach its face specified by the second argument
- `void rotate(int module_id, int servo_id, int angle)`: Tells the module, whose ID is specified by the first argument, to rotate its servo specified by the second argument by the angle, in degrees, specified by the third argument
- `void is_attached(int module_id)`: Returns only when the module, whose ID is specified by the argument, has acknowledged its attachment
- `void is_detached(int module_id)`: Returns only when the module, whose ID is specified by the argument, has acknowledged its detachment
- `void has_rotated(int module_id)`: Returns only when the module, whose ID is specified by the argument, has acknowledged its rotation

Remarks :

1. The third argument of the functions `attach` and `detach` is used only for Cube2 and is the one equivalent to the `Behaviors` argument in the decentralized controller :
   - 0 : `WITH`
   - 1 : `WITHOUT`
   - 2 : `ONLY`

3.2.1 The Sequence Protocol

In order to ease the use of the Supervisor and to create reconfiguration sequences more easily, the Supervisor is equipped with a file parser. These files must be placed in the Supervisor's directory, must be plain text and must be written using the small protocol I have created for this. To make the Supervisor read a file, you simply have to write the full name of the file in the `controllerArgs` field which is in the `Supervisor` node in Webots. You can make it read various files, one after the other, by placing their names in the right order in the field and seperate them with white spaces.

The protocol of the files is very simple :

1. A line starting with the "#" character is a comment line
2. All the other lines, which contain more than one character, must be commands

The following commands have been implemented :

- `A module_id face_id [behavior]`: Attaches the face *face_id* of *module_id*
- `D module_id face_id [behavior]`: Detaches the face *face_id* of *module_id*
- `R module_id servo_id value`: Rotates the servo *servo_id* of *module_id* by *value* (an integer, in degrees)
- `N [angle]`: Places a new module on the initial position with a rotation of *angle*, in degrees, on the Y axis
- `O text`: Outputs on the log *text* (no need for "")
- `a module_id`: Waits until *module_id* has attached
- `d module_id`: Waits until *module_id* has detached
- `r module_id`: Waits until *module_id* has rotated

Remarks :

1. The `N` command represents a new module reaching the reconfiguration site. The `angle` argument is needed in order to allow the module to start in any position
2. The argument *behavior* is used only for Cube2

3. The arguments enclosed in square brackets are optional arguments: in case they are not specified, the default value 0 is used

4. The specification of the protocol, along with an example of use, can be found in the file `sequence_protocol.txt` which is located in the controller's directory of the Supervisor

Chapter 4

Results

Using everything I developped for this project, I was able to analyze the fitness of the modules and find macro-movements for the most promising one. I analyzed only Cube1, Cube2 and Cube3 as the two remaining ones had some drawbacks explained in 2.4 and 2.5.

4.1 Modules' Fitness

In order to analyze the fitness of the modules, I had to choose a metric that would allow me to compare them. After some discussions agreed on the following measurement :

$$cost = servos \cdot actions \tag{4.1}$$

where *servos* is the number of used servos, we do not count them more than once if they are used more, and where *actions* is the total number of rotations done by servos in the sequence. The lower the *cost* is, the better the result is.

Even if it looks simple, this metric gives a good representation of what we are searching in a good module :

- Neither the organisation nor the number of servos on a module matter: the important characteristic is that the module uses few of them
- Rotating for a small angle or for a big angle does not change anything, we want to rotate as few times as possible

Module	Passive Structure						Active Structure					
	T1	T2	T3	T4	T5	Total cost	T1	T2	T3	T4	T5	Total cost
Cube1	2	1	1	X	X	4	10	1	1	16	12	40
Cube2	4	80	80	0	2	166	20	2	2	0	15.5	39.5
Cube3	2	1	1	X	44	48	10	1	1	16	12	40

Table 4.1: The fitness results

Rank	Passive Structure	Active Structure
1.	Cube1	Cube2
2.	Cube3	Cube1
3.	Cube2	Cube3

Table 4.2: Ranking of the modules

As what we wanted to test was the fitness of a module for self-reconfiguration, I decided to test each module on all the movements it would need for reconfiguration. I found five basic movements which are labelled as follows :

T1 The module should move straight forward

T2 The module should turn 90° left

T3 The module should turn 90° right

T4 The module should turn 90° up

T5 The module should turn 90° down

I also tested two kinds of environments that reflect the possible ones the modules could encounter :

Passive Structure The modules are moving on a modified ground which does not have any servo but have some Connectors , so they have to move totally by themselves

Active Structure The modules are moving on a structure made out of other modules that help them as much as possible

The main idea of these tests was to find the smallest group of modules, or the structure organization, that allows them to perform all these tests. These configurations, and the structure organization, are represented in subsections 6.2.1 and 6.2.2 respectively, in the appendix.

In the table 4.1 the results of each module for every test are listed. An "X" symbol in a cell means that the module was unable to perform this movement. Using these results we can rank them, as shown in table 4.2.

Remarks :

1. It is important to notice that even if Cube1 and Cube3 have better results in Passive Structures, they did not succeed in all the movements. It is interesting to notice that, even if this looks like a drawback, this reduces the number of possible paths when reconfigurating which could be interesting
2. On the Passive Structure, Cube1 is marked as unable to perform T5 because it would have needed too many modules in order to achieve it and this would not have been realizable using real modules
3. In order to achieve T4, Cube1 and Cube3 need the help of a specific part of the structure I called "loading module"; its configuration can be found in subsection 6.2.3 of the appendix
4. On the active Structure, Cube1 and Cube3 have exactly the same fitness. I placed Cube1 over Cube3 because its module is simpler than the one of Cube3

4.2 Cube2 Macro-movements

For reconfiguration, the main goal is to place every module at the useful position on the structure. In order to achieve that, the modules need to move over the structure itself. As this structure is made out of other modules, it is what I called in my tests an Active Structure. For me, Cube2 is the most promising module as it has the best results on Active Structures. Moreover, it is the only one that is able to perform all the movements on Passive ones too.

In order to find macro-movements, we tried to create a piece of furniture with this type of modules. Using the sequence of movements needed by the modules to get in place, I was able to isolate totally five macro-movements :

M1 Move on our own to a side module (Fig. 4.1)

M2 Move on our own from a face to another one (Fig. 4.2)

M3 Move another module from a side module onto ourselves (Fig. 4.3)

M4 Move another module from ourself onto a side module (Fig. 4.4)

M5 Move another module from a face to another one (Fig. 4.5)

The pseudo-codes corresponding to these movements are available in table 4.3. Note that in these figures :

- The dashed cube is the moving module
- The green cube is the module that is using its lock
- The red L-shape is the used lock
- The red arrow is the movement of the lock
- The black arrow is the movement of the module

Remarks about the macro-movements :

1. The sign of the rotation can change depending on the orientation of the module
2. `Ensure all are connected` means that we need to reconnect all the Connectors which have be disconnected during the macro-movement
3. The commands given here in pseudo-code are macro-commands that could need to be replaced by more than one "real" command

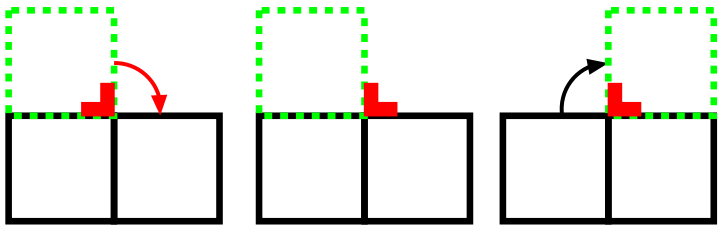

Figure 4.1: The macro-movement M1

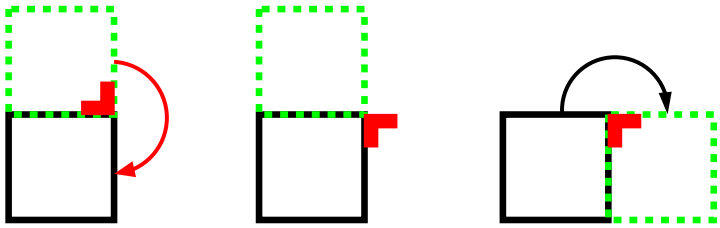

Figure 4.2: The macro-movement M2

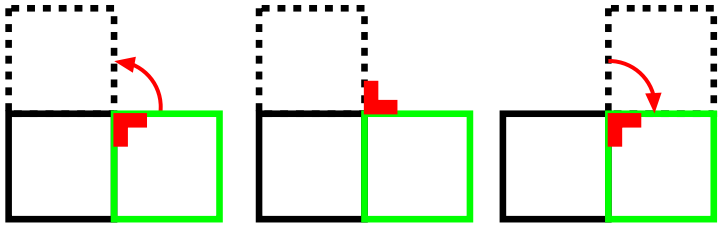

Figure 4.3: The macro-movement M3

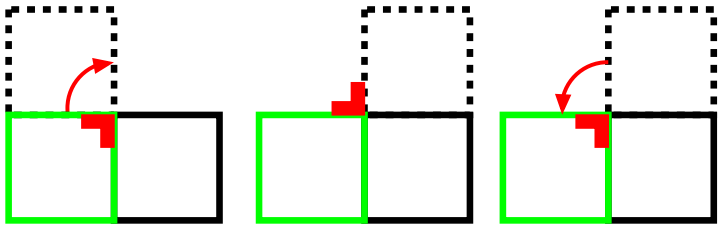

Figure 4.4: The macro-movement M4

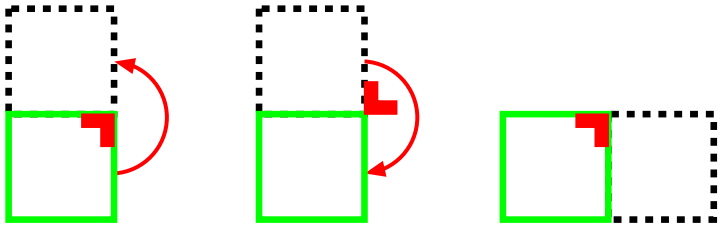

Figure 4.5: The macro-movement M5

M1	**M2**
Free the lock	Free the lock
Rotate 90 °	Rotate 180 °
Wait	Wait
Attach the lock	Attach the lock
Detach the module	Detach the module
Rotate −90 °	Rotate −180 °
Wait	Wait
Attach the module	Attach the module
Ensure all are connected	Ensure all are connected

M3	**M4**	**M5**
Free the lock	Free the lock	Free the lock
Rotate 90 °	Attach the lock	Rotate 180 °
Wait	Wait	Wait
Attach the lock	Detach the module	Attach the lock
Wait	Rotate 90 °	Wait
Detach the module	Wait	Detach the module
Rotate −90 °	Attach the module	Rotate −180 °
Wait	Detach the lock	Wait
Attach the module	Rotate −90 °	Attach the module
Ensure all are connected	Wait	Ensure all are connected
	Ensure all are connected	

Table 4.3: The macro-movements in pseudo-code

Chapter 5

Conclusion

The main goal of this project was to develop a new module that could be used for reconfiguration in the Roombots project. As I was starting from scratch, I created the simulated models of the possible modules, I implemented the necessary controllers and I elaborated the tests used to measure the module's fitness for self-reconfiguration.

During this project and using the tests I made, I was able to first remove the unsuitable modules and then rank the useful ones. It gave me a clear idea about what the advantages and the drawbacks of each module were. It allowed me to select the more promising model and to try a centralized controller on it. Cube2 looks very promising even if it might be technically more difficult to build.

I had also the surprise to notice that in the case of Cube3, adding more degrees of freedom does not increase the capabilities of the module with respect to the ones of Cube1.

I hope the work I did will be useful and will ease the use of these modules in the Roombots project. I am confident in the fact that finding macro-movements like I did for Cube2 is useful in order to find the reconfiguration sequences. I also think that the tools I have created will allow everyone to find the macro-movements for the remaining modules and new ones too.

Chapter 6

Appendix

6.1 Organization Of The Files

All the work I have done during this semester project can be found on the attached CD. It is also on the web page of the BIRG's group in the projects branch. This file are organized as on figure 6.1.

- The folder `movements_active_structure` contains all the files used for the tests T1 to T5 on an Active Structure, except the ones for Cube3 as its movements are exactly the same as the ones of Cube1. T1 for Cube3 is included as an example
- The folder `movements_passive_structure` contains all the files used for the tests T1 to T5 on a Passive Structure, except the files corresponding to the tests a module did not achieve
- The folder `removed_models` contains the Webots files for the models of Cube4 and Cube5
- The folder `roombots_centralized` contains the Webots files and the centralized controllers for the models Cube1, Cube2 and Cube3, along with the controller for the Supervisor and the `sequence_protocol.txt` file
- The folder `roombots_decentralized` contains the Webots files and the decentralized controllers for the models Cube1, Cube2 and Cube3
- The folder `roombots_stool_example` contains the Webots files and the centralized controllers used to create a basic stool using Cube2

modules. These are the files I used to find the macro-movements of the Cube2 module

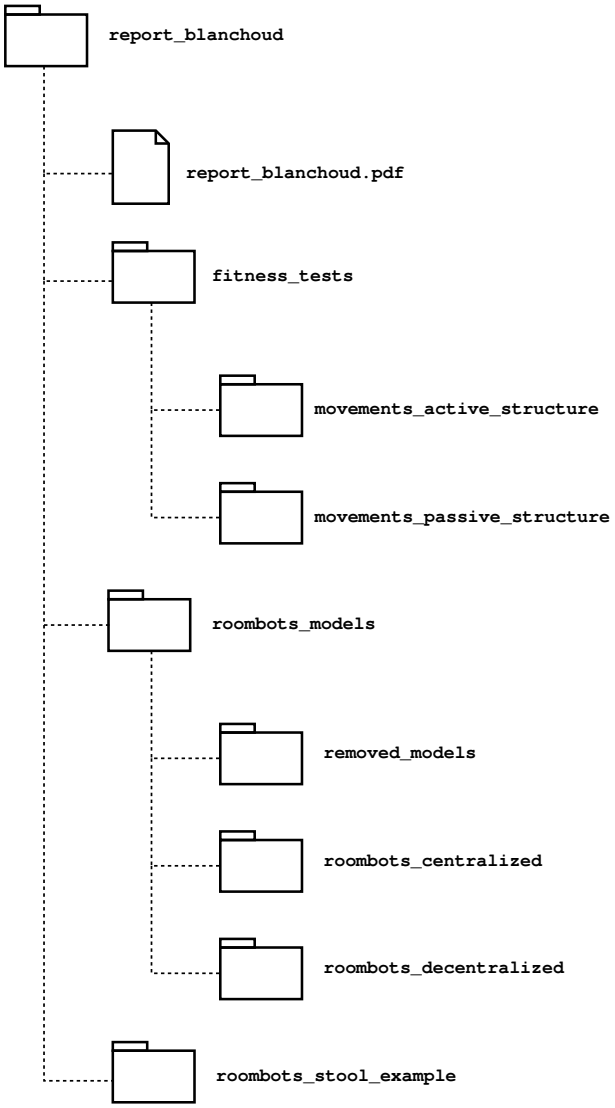

Figure 6.1: The organization of my files

6.2 Modules' Configurations

In all the figures of this section, I follow the Webots color convention for the axes orientation :

X : Red

Y : Green

Z : Blue

6.2.1 Smallest Moving Group

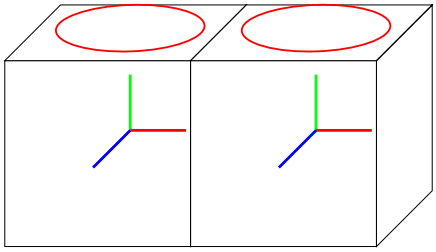

Figure 6.2: The Cube1's smallest group

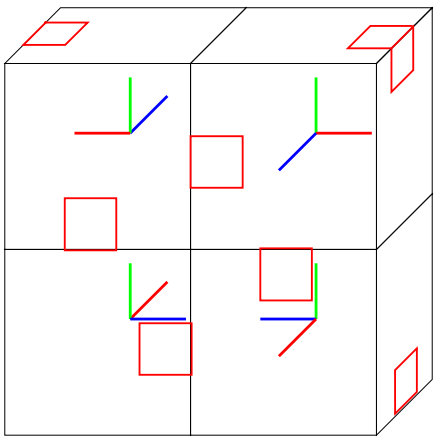

Figure 6.3: The Cube2's smallest group

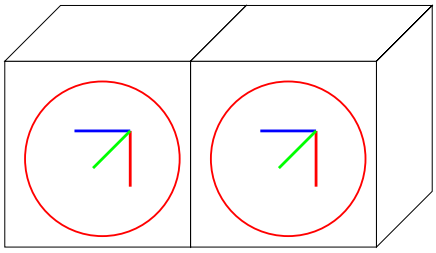

Figure 6.4: The Cube3's smallest group

6.2.2 Structure Organization

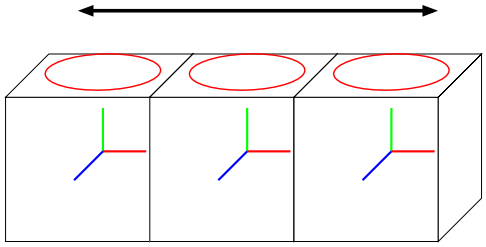

Figure 6.5: The Cube1's structure organization

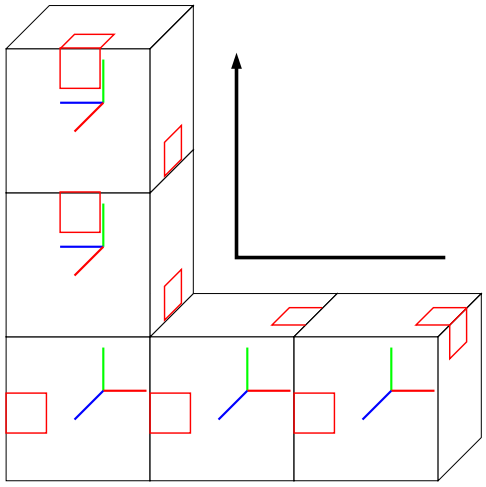

Figure 6.6: The Cube2's structure organization

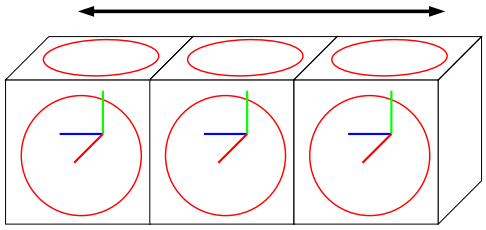

Figure 6.7: The Cube3's structure organization

6.2.3 Loading Module

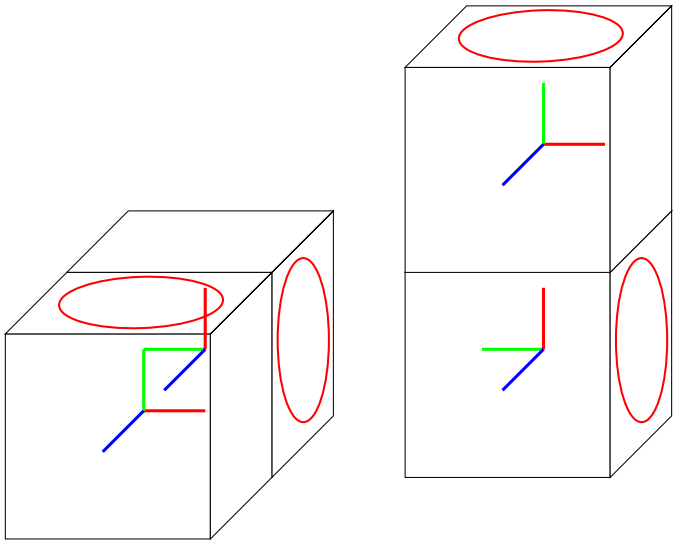

Figure 6.8: On the left, the loading module when moving up; on the right, the loading module when moving down

The loading modules must be attached to the structure either on face 0 on of face 5 of the "base" module. The "base" module is the back one for the up-loading module, and is the lower one for the down-loading module.

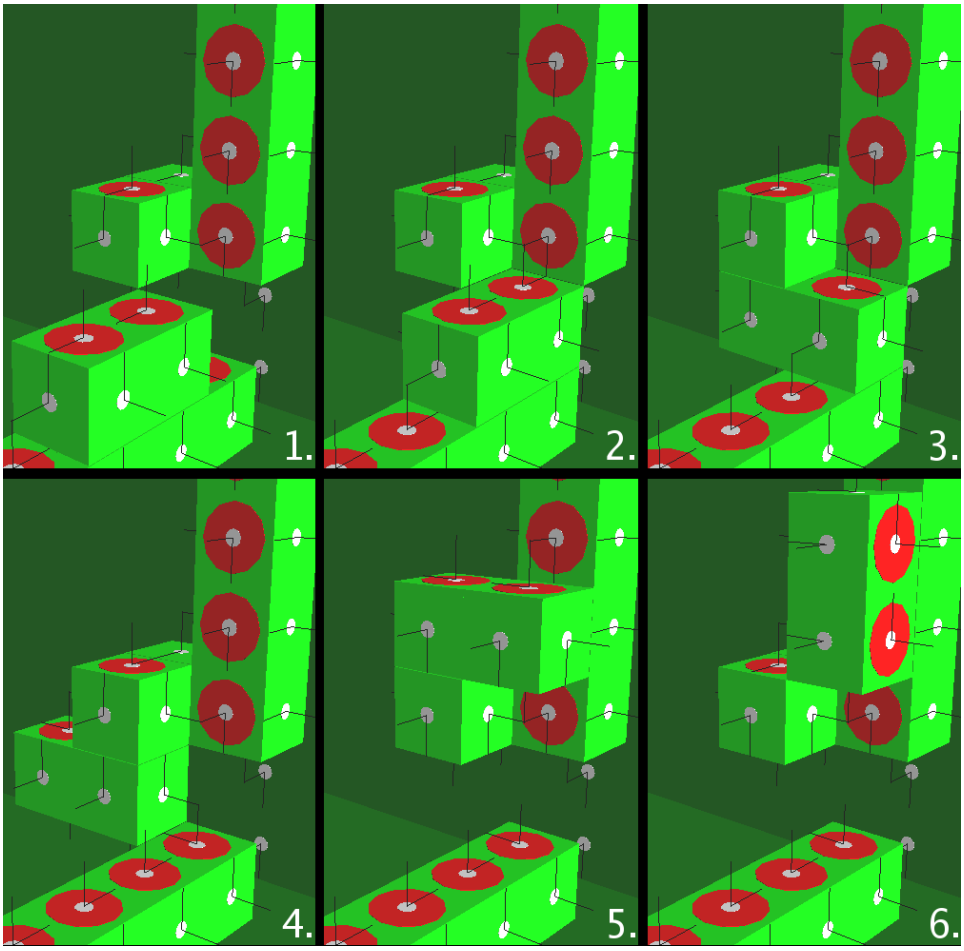

Figure 6.9: The loading sequence when moving up

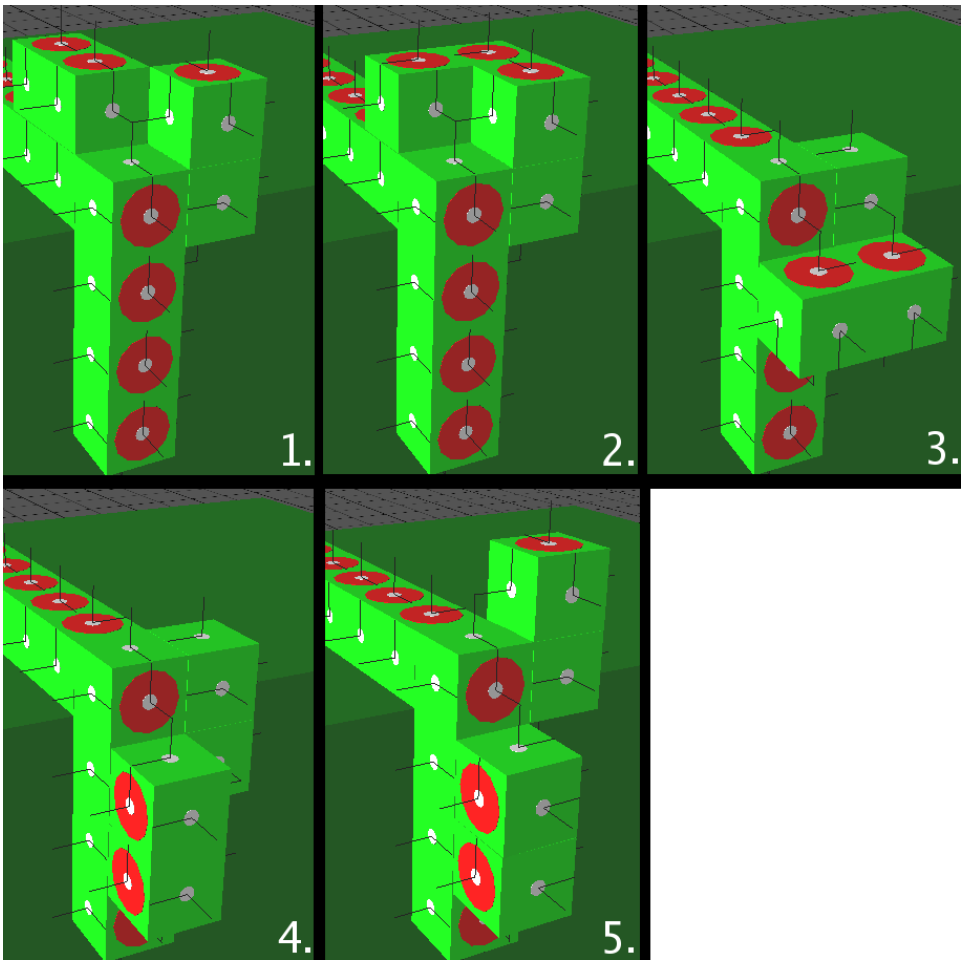

Figure 6.10: The loading sequence when moving down