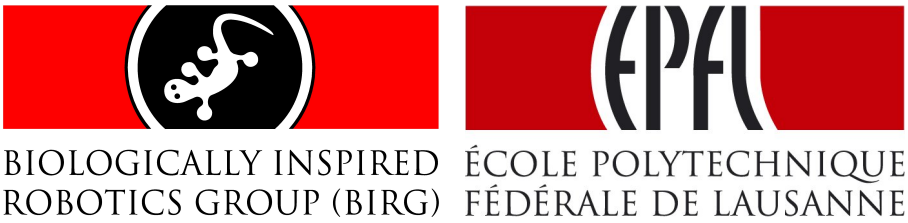
École Polytechnique Fédérale de Lausanne
Section de Microtechnique
TULEU Alexandre



Semester Project Report

Roombots - Central Pattern Generators, symmetries and online learning

Professor : Auke Jan Ijspeert
Supervisor : Alexander Spröewitz

Abstract

This semester project focus on the problem of locomotion in Modular Robotics. This project is a continuation of the work of several former Biologically Inspired Robotic Group (BIRG) students on this topic. Since modular robots are made of several identical modules, that can assemble each other in almost an infinite number of configurations, these works give priority applying a Central Pattern Generator (CPG) approach, rather than a more classical one, where a fine model of the robot kinematics and dynamics is needed.

The Central Pattern Generators models are well known in Biologicaly field, to be responsible of the generation of rythmics mouvement in vertebrate. Most of them can modulate the pattern created just by acting on a few parameter, and therefore, they are powerful tool to encode the movements.

Former works has lead to methods were, given a certain modular robot, we manually design a CPG for it, and then run online optimization algorithm, in order to make the robot learn an optimal gait.

Most of the former work was made on the YaMoR modular plateform, and have to be updated on a new one, currently develloped at BIRG : the Roombots modules. Indeed, this module add new possibility, as self-reconfiguration, and therefore we want the Roombots robots to be able to automatically determine their CPG. Making this task to still be computed online is a challenging problem.

In this document, we will first describe some software tools we made for manipulating Central Pattern Generators, Optimization Algorithms, and make the Roombots simulation more accurate, in enhancing the collision detection in the Webots simulation software. Then we present a study of the locomotion of a specific Roombots robots, and then we propose an algorithm for automatically design CPG model for modular robot.

Contents

Chapter 1

State of the Art

1.1 Modular Robot that adapt themself to their environnement.

Nowaday most of the existant robots (especially industrial robot), only operates in fully controlled environnement, in order to avoid unexpected interaction with unknown agents. Such a strategy is no longer feasible when we want robot that can interact with human, in our daily environnement. So the design of polyvalent and adaptive robots is one of the main challenges in Robotic field. In order to adress this problem, one approach would be to imitate human behavior. As humans have continually transform their environnement to fit better their capacity, Humanoïd robot will then have more capacity to evolve with humans.

Another way is to go further the most adaptability and polyvalent form. One idea is to have a modular, self-reconfigurable approach. Like cells in animal organism, a whole robot will be made of several small identical parts, called module. Each module will not be able to do much own his own, but when inderacting with other modules, can do some desired tasks.

This approach has put the adaptibility on top of their priority, as the module are *self-reconfigurable*, i.e. they can dissasemble and reassemble to form new morphology, that are better suited for a certain task. For example a legged robot, blocked by wall with a small hole on it, can reconfigure as a snake-like robot, go through the hole, and then reassemble as a legged robot afterwards.

This approach will also lead to economical benefit. They will be able to adress a lot of different tasks, by using the same hardware, leading to scale effect reducing cost in production.

1.1.1 An historic of modular robot.

The first modular robot is believed to be CEBOT (handling for CEllular roBOT), created by Fukuda et al [FBHK91]. Since, over more 30 system have been made ([Wik]),as M-TRAN[ST08], Polybot [Cen97], Molecube [ZCL07], and YaMoR, previously develloped at BIRG ([Yer07] [May07]). According to [Wik], the Modular robots can at least be classified by two properties : their geometrical architecture (Lattice/Chain), and the way they reconfigure (deterministic/stochastic) :

(a) the ATRON modules : a Lattice architecture

(b) the YaMoR modules : a chain architecture

Figure 1.1: The two type of geometrical architecture in Robotics

- Lattice architectures have units that are arranged and connected in some regular, space-filling three-dimensional pattern, such as a cubical or hexagonal grid. Control and motion are executed in parallel. Lattice architectures usually offer simpler computational representation that can be more easily scaled to complex systems.

- Chain architectures have units organized and connected on a tree or graph topology. Control and motion are often supervised and centralized. These architecture is more versatile, but is harder to compute and analyse.

- Deterministic reconfiguration, when the position of each unit is known at each time and modules are moving directly to their target during reconfiguration.

- Stochastic reconfiguration, where the movement of module follow a stochastic process. Therefore the reconfiguration is only statistically guaranted.

One can notice, as more and more Modular robots have hybrids capacity, the distinction between chain and lattice has become less and less important.

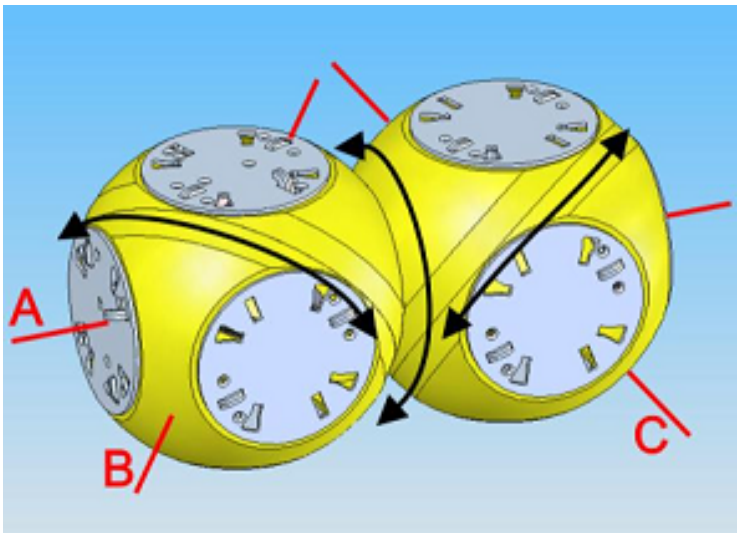


Figure 1.2: Design of one Roombots module.

1.1.2 Introducing the Roombots.

The Biologically Inspired Robotic Group (BIRG) is currently designing a new Modular Robot, called Roombots. This project is the sucessor of the two former project YaMoR and DoF Box. These two projects were pure chain-type modulars robots, that can only be manually configured. One of the key feature that add Roombots to these two projects is the capacity of the module to dynamically connect them [SABI08], and lets the Roombots module to perform *self-reconfiguration*. The main goal of this robot is to make intelligent domestic furnitures, which explains its name, as it will stay inside a domestic area. This module is bigger than the previous on made at BIRG, and have three degrees of freedom, allowing with two modules put in chain, to reach any arbitrary position [May09].

1.1.3 The locomotion problem in Modular Robotics.

In order to evolve in a non controlled environnement, a key characteristic of robots is the ability to move. In modular robotics this problem is adressed in two way, according the fact we prefer an lattice architecture or a chain/tree architecture.

Lattice modular robot will mainly move by continously reconfigure them. Each module will move on the structure, and then create a displacement of the whole structure (see figure 1.3). Making controller able to perform this task, and that are scalables, is a challengic topic in Modular Robotic. You can see work on this topic in [VKR08].

Chain type robot ,on the other hand, have a fixed structure and then the problem is much closer from classic robot Locomotion, since we want to find the best way to use our actuators, in order to get an desired displacement. On classical robot, this work is often done by inverse Kinematics/Dynamics, and performing optimization (see [CMLA07] for an example of such a controller).

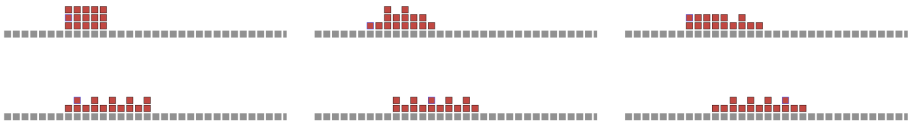


Figure 1.3: An example of locomotion for lattice type modular robot taken from [VKR08]. Each red square represent a module.

But here, in modular robotics, as the avalaible configurations for a robot are close of infinity, it is better to find a systematical way to perform locomotion.

There is several former work done at BIRG on Locomotion in modular robotics, the oldest one begins on the YaMoR project. All of them, have mainly the same apporach : the use of an Central Pattern Generator (CPG) model, as a way to encode and to control the movement of the structure. Then an optimization is performed on the parameter of this model, in order to make our Modular Robot learn its gaits.

As a learning process, this optimization can be either computed offline, i.e. by running all the test in a computer, in an simulated physic environnement. By this way, we can run the test faster, as they can be computed faster than real time, and some distribuable algorithm let us repartize the computation time among computers. Once the optimum reached, the value are just brought back to the real system

On the other hand, this optimization can be run online, i.e. by testing directly on the hardware, at a real time. This methods let us to do some longlife learning : the system doesn't stop the optimization step. This approach is more robust, as it able the robot to adapt to environnemental condition changes. For example, a change of friction coefficient with the floor, can move dramastically the optimal gait. A robot performing longlife learning will be able to learn this nw gait. Furthermore, lifelong learning accord well with modular robotics, as the robots can reconfigure them.

1.2 Previous work done at BIRG on locomotion.

Before to go further, we want to introduce some key concept that use the former BIRG project to adress the problem of modular robots locomotion.

1.2.1 A brief review on Central Pattern generators.[1]

Central Pattern Generators is now a well diffused assumptions in biologically fields, as mentionned in [Ijs08a], that movement in animals are *centrally* generated through block of neurons, that does not require sensory-feedback information from the perephirical neural system. Both discrete and rythmic moves are created in an open-loop process.

Here, as our study is on the locomotion, we will only talk here about rythmic Central Pattern Generator models. There are some experiment that have shown

[1] This section provides a really quick review on CPG, mainly inspired by [Ijs08a], for further details, please refers to the main paper.

that CPG are made from several coupled oscillatory neuron centers. Then a rythmic CPG, could be modeled as a network of coupled Oscillator.

CPG is a big field in bilogically research. The experiment made, have brought to etablish several Neurobiological models, at different levels. All these model are good tools for describing the movements, and there is more and more research over the past few year, that try to use this model in order to control the locomotion of robots. [Ijs08a] propose up to five reasons for prefering the use of CPG instead of classical methods such as inverse kinematics/dynamics or the control of criterion like ZMP ([KKK+03],[Wie06]) or Capture Point [PCDG06] :

1. The purpose of CPG models is to exhibit limit cycle behavior, therefore they are more robust to perturbation.
2. They are well suited for distributed implementation, and that is very intresting for modular robots.
3. CPG models have only few parameters to drive speed, direction or even the type of gait. Therefore, at higher level, the command is simplified, as command just have to provide anly a small number of variable.
4. CPG are ideally suited for sensory feedback, that able us to control finelly the mutual interaction between the CPG and the body dynamics.
5. CPG are a really good substrate for learning and optimization algorithm.

Now we will present more formally two types of oscillator, that can be taken to design a CPG model. Note, as an anology with biological study, we will also name the oscillators neurons, as in biological fields, these oscillators are just a mathemical macroscopic model of how a group of neurons act.

Amplitude driven oscillators with direct phase coupling

These oscillators are almost the simplest oscilattors we can use for a CPG. Such a network could be made of n oscillators, defined by the above differential equations :

$$\dot{\phi}_i = \omega_i + \sum_j \omega_{i,j}.r_j.\sin(\phi_j - \phi_i - \psi_{i,j}) \tag{1.1}$$

$$\ddot{r}_i = a_r\left(\frac{a_r}{4}(R_i - r_i) - \dot{r}_i\right) \tag{1.2}$$

$$\ddot{x}_i = a_x\left(\frac{a_x}{4}(X_i - x_i) - \dot{x}_i\right) \tag{1.3}$$

$$\Theta_i = x_i + r_i.\cos(\phi_i) \tag{1.4}$$

where Θ_i is the output of the oscillator, ϕ_i its phase, r_i its amplitude, x_i its offset. Equations 1.2 and 1.3. are to let the amplitu and the offset value, only have smooth changes. ω_i is the intern frequency of the oscillator. It will be the frequency of its output when the oscillator is not coupled. Equation 1.1 show some coupling termes, between neuron j and i, where $w_{i,j}$ is the weight, and $\psi_{i,j}$ is the phase bias of the directed coupling from neuron j to neuron i.

According to [Ijs08b] such an isolated oscillator have lot of interesting behavior. First of all, if we put on some special case, such as all the oscillator to

have the same inner frequency, and that all bi-directional coupling are choosen to be opposite, then in most case, our network will phase lock. In some case, such in the case of a chain of ascillator, a criteria for such a behavior can be found.

Hopf oscillator

Another interesant CPG model proposed in [RI08], is one made of coupled Hopf oscillators. Such ocillators are driven by the following differential equations.

$$\dot{x}_i = \alpha\left(\mu - r_i^2\right).x_i - \omega_i.y_i \tag{1.5}$$

$$\dot{y}_i = \beta\left(\mu - r_i^2\right).y_i + \omega_i.x_i + \sum_j k_{ij}y_j \tag{1.6}$$

$$\omega_i = \frac{\omega_{stance}^i}{e^{-by}+1} + \frac{\omega_{swing}^i}{e^{by}+1} \tag{1.7}$$

According to [RI08], this oscillators exhibit a limit cycle, such as both variable x_i and y_i will have oscillations of amplitude $\sqrt{\mu}$. But what another feature that is bring with this model is the possibility to choose independantly the swing and stance frequency, i.e. we want the oscillation not to be symmetric, and faster while increasing than when decreasing. [RI08] also explain how to design a CPG for a quadruped robot with this model, and how we can put some sensory feedback in the oscillators.

1.2.2 A review on the former locomotion project.

There has been several former projects, over the past few years, on the locomotion in modular robotics at BIRG. We present here Some key results of this studies.

First of all, all this project point up the utility of the Central Pattern Generators. Their main idea for almost of them ([May07], [Wie08] [Lé08], [Yer07], [Bou04] [Mar05], ...) to manually design a Central Pattern Generator, for some given assembly of modular robots, and then to run on optimization algorithm, in order to determine the parameter that best fit one fixed criterion, which most of time is the covered distance by the robot during a fixed amount of time.

Some one key results that appears of this work, is that for most cases, this criterion is multimodal. Then our optimization algorithm should be enough robust, not to fall in local extrema.

One of the key ideas devellop by all authors, were to put online optimization as a goal, for the study which were limited to simulation ([Wie08], [Lé08], [Bou04], [Mar05], ...), or as a fact when acting on real robot. In order to do this, they classified several optimization algorithms, in order to determine which one will be the best suited for this task ([Bou04], [May07]), and its appears that for YaMor modules, the most performant algorithm is the Powell's methods [May07].

Most of this study was made on YaMoR robots, which modules have only one degree of freedom, and were not able to self-reconfigure them. The passage to the new Roombots modules, which have this capacity and own three degrees of freedom (DoF) brings new problems :

- A problem of scale, since the size of the CPG network increase significantly with the new added DoF. Then the CPG has to be more carefully design, if we still want to be able to perform online Optimization. One relevant idea brought by [Wie08], is to use the symmetries in order to reduce the search space.

- As the ROOMBOTS robots can self-reconfigure, the manual design of the CPG network is no more a solution. We have to automatize as much as possible the CPG design processus.

- Strangly, the Powell's Methods seems to be less robust with the ROOMBOTS modules than with YAMOR one, and the optimization process process often fall into local extrema according to [Wie08]. Then we have to look for new optimization algorithms.

1.2.3 Goals of our project.

Then it appears that our main goal in this study are :

1. An extensive literature research about Modular Robots, Self-Reconfiguration Modular Robots, symmetries, and (online) optimization methods.

2. Implement several RB robot configurations in Webots, and find ways (automated if possible) to check for morphological symmetries (e.g. graph representation of the mechanical structure [ASB+08]). Use those symmetries to distribute CPG parameters while applying the optimization.

3. Test and select different CPG models, e.g. [IC07], [RI08]. Come up with a method for this (evualuation matrix etc. . .).

4. Test and select different optimization methods and their corresponding cost functions.

5. Optional: life-long-learning. Including sensor feedback from the ROOMBOTS modules into the CPG.

Chapter 2

Tool Creation

One of the major part of my work was the creation of the tool for manipulating ROOMBOTS in Webots. Although there was previous works on simular subjects ([Yer07] [Lé08] [Wie08]), there was no generalized framework provided, including works done on different languages (C, C++ C#). Therefor the whole software has been redone from scratch. C++ language has been choosen, for its modularity possibilities as Object-Oriented Language and for its cross-platform capacity, being available on major platforms (Windows, Linux and Mac OS X). For some very specific task, like XML parsing and Network capacity, Qt Libraries were used, well-knowed as good Open Source Cross Platform libraries for making complex applications. All the code were successfully compiled and executed on both Linux (Ubuntu 8.04) and Mac OS X (10.4 and 10.5), transition to Windows plateform could be easily made through the use of Qt tools.

As the first goal of our study we wanted to test several Central Pattern Generators (CPG) on several ROOMBOTS Structures, all the controllers have been designed to be configured by an XML configuration file. This file describes :

- The way the ROOMBOTS are assembled and initially positionned in the simulation.
- The way the CPG is designed and the way Neurons drive the structure.
- Which parameters of the Network are open for optimization, and which are fixed.
- the parameters and the choice of an Optimization Algorithm, to train the Neuron Network, if wanted.

The Syntax of this file is described in Appendix A. All the low-level documentation as been written inline in the source code in Doxygen format. Here, we will just see the libraries at a global Scope.

2.1 libCPG : a library to manipulate Central Pattern Generator.

Since the first goal was to test different Central Pattern Generators on several ROOMBOTS Structure, we decided to make a general library to create rythmic Central Pattern Generator (CPG), as a network of coupled non-linear oscillators. This library has been made to help the develloper to separate the inner way of how a CPG works, and how we can use it to drive a structure. It allows to change the type of CPG used to drive our robots without having to recompile the code, just by parsing an XML file. Allthough the main purpose of this library is to provide several CPG model to the user, only one has been implemented. This model is a network of amplitude-driven, phase oscillators, with direct phase coupling [IC07].

2.1.1 General Organization of the classes

A rythmic CPG can globaly be seen as Network of coupled oscillators. Since we want the libCPG to be able to be distributed in several program, we don't choose the simpliest way to represent the graph by vector and coupling matrix, but with 4 main objects CPG::Neuron, CPG::State, CPG::Link, and CPG::Param:

- the CPG::Neuron is the main object of the network. They compute at each time step the differential equation of the oscillators, and share CPG::State throught CPG::Link.
- the CPG::State represents the state of a Neuron, i.e. it stores the Neuron inner variables, which must be shared with other.
- the CPG::Link describes how two neurons are connected. There are only directed Links, from a parent to a child. This object exists only because some CPG types (as Phase-based network) need several variables for one Link and other only one (as Hopf oscillators).
- the CPG::Param describes the parameter of a Neuron.

The way these classes work together is summed up in the figure 2.1.

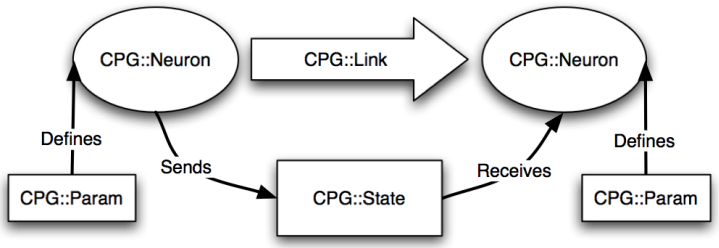


Figure 2.1: General Organization of the libCPG main classes

Serialization capacity.

Since we want to distribute the network among several programs, CPG::Link, CPG::State, and CPG::Param classes are able to serialize them in an array of char, in order to be sent through Webots Emitter / Receiver that simulate a serial radio connection. All this work has been done with the help of the Mikaël Mayer's library libMessage (see [May09]).

Managing the CPG::State

In order to work, the Neurons have to share their states at each time step. But since we use inheritance mechanism to write generic controllers, wich only use abstract classes, we can only use pointer and manage carefully the memory. For CPG::State, wich must be created and destroyed at each timestep, it is very difficult to achieve this task. Two classes, CPG::LocalStateStorer and CPG::StateStorer have been created in order to manage, respectively, CPG::State wich are shared between Neuron on the same program, and CPG::State received from another program through Message.

2.1.2 Implementation in a Webots controller

This library was implemented in two Webots controllers : CPGController and CPGSupervisor.

The role of CPGController, is to drive a ROOMBOTS module, and share with other CPGController the CPG::State of his interns neurons. The CPGSupervisor, is a more complex program, that parses the XML file, and drive the other CPGController :

- It sets up the CPG among the module, send them order to start or stop their neurons, change the parameters of the network...
- It can either run the simulation indefinitly, using the parameters provided by the XML file, or run an optimization algorithm to train the CPG, or either run as a client for a OptimizerServer see section 2.3.

The figure 2.2that sum up this organization :

2.2 Enhancement of the geometrical model of ROOMBOTS in Webots.

Another tool that has been made for manipulating ROOMBOTS in Webots, is the enhancemend of the collision detection. Indeed, the ROOMBOTS have a particular design made of four part, each can be seen as a half-sphere truncated with three plane (see figure 2.3).

Open Dynamics Engine (ODE - the physic engine running behind Webots) can only perform collision detection between primitive volumes (i.e. sphere, box, cylinder, ...) and we can only make union of these primitives, in order to make more complex shapes. Our main problem is that the particular shape of ROOMBOTS can not be approximated by an union of primitives.

To fix this problem, the ROOMBOTS model in Webots only takes two Spheres in order to approximate the shape of one ROOMBOTS module. This lead to some

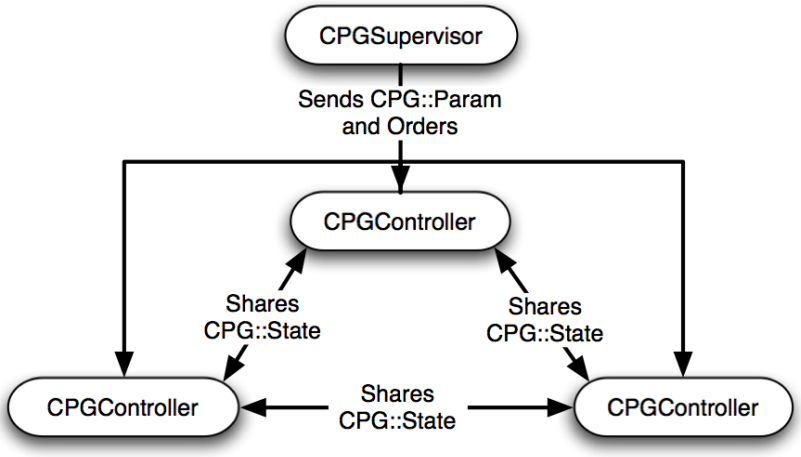


Figure 2.2: General organization of the Webots controllers.

limitations in the use of the ROOMBOTS module in simulation : one module can not stand vertically on a flat surface, as it can only have a single point of contact on it. Some simulations don't have feasible motion, as modules can roll in case they should not be able to (see the car simulation result of [Lé08])...

2.2.1 Creating a particular dGeomClass for ROOMBOTS modules.

The problem described above can be solved in two way : Use the trimesh used by Webots for displaying the modules, and ask for Webots to use this shape. Even if this solution is easy to put in place, it is not envisageable, since it will lead to a huge amount of computation in physics simulation, and decrease significantly the Webots performances.

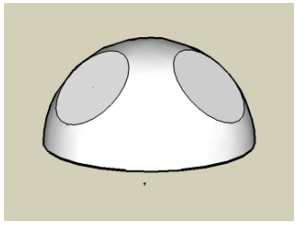


Figure 2.3: The dRoombotShape the basic part of a ROOMBOTS module.

Another way is to provide to ODE a new primitive (a dGeomClass in Webots API), that wich exactly the ROOMBOTS shape, and then provides a collision detection function of this class with each other ODE primitive classes. This leads to a more precise approach, and don't increase much the computation time (as much as replacing the sphere by a box). In our cases, for most of primitives, it is not an easy work, but in the case of collision with an infinite plane, the solution is approachable.

This new dGeomClass, called dRoombotShape (see figure 2.3) has been created, and only the detection with infinite plane has been provided, letting us to make better gait simulation experiments, but disabling the capacity of webots :

- to detect collision between modules.
- to manage the interactions with other objects than the ground.

2.2.2 Detection of the collision with a plane.

Here we present how we compute the collision detection for the dRoombotShape with an infinte plane.

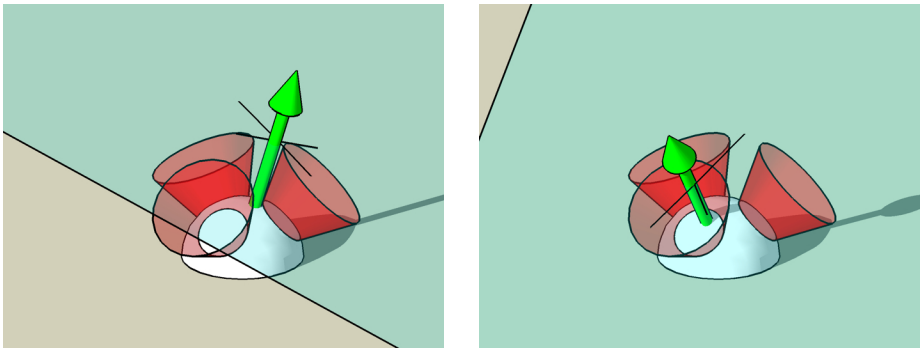
Functionnement of ODE detcetion collision

When calculating the interaction between two solids (contacts forces), ODE need to know where it can apply these efforts on both Solids. The main goal of the collision detection is to find these points, and their associated normals of contacts, i.e. the direction in which he could apply these efforts (at a first approximation, when there is no friction between the two solid, in other case, there could be an tangential effort). ODE, also needs to know the depth of the contact points, i.e. the algebric distance of interpenetration, along the normal axis of the contacts, beeing positive if solids collide.

In order not to hugely increase the computation time of the physics constraint resolution (the step after the computation), we have to keep this number of contact as low as possible.

Description of the algorithm

In order to determine this contact points, ODE let all the solids slightly interpenetrate each other. Then the point(s) returned must be the deepest point(s) of the dRoombotShape volume, which is on the other side of the plane. We can easily determine this point, by computing the projected P of the center of the half sphere C on the plane. then the direction of the best candidate is given by the vector $\vec{CP}$.



(a) best candidate lying on a spherical part (b) best candidate lying on a flat part

Figure 2.4: Representation of the two major cases hapening while determining the best potential contact point on a dRoombotShape.

Then two cases could happen :

- The best candidate lies on a spherical part of the shape(see figure 2.4a). Then, we just found the potentially point of contact (we must then verify that this point interpenetatre the half space defined by the plane).

- The best candidate lies on a flat part (see figure 2.4b). Then another computation is needed to determine whether the shape has to lie on the whole flat part (and then we send back only 4 contact points that are already precomputed) or if the contact is made on the frontier between flat and spherical part, which is a circle.

To determine this last case, we compute the depth of the deepest and the least deep point on the circle :

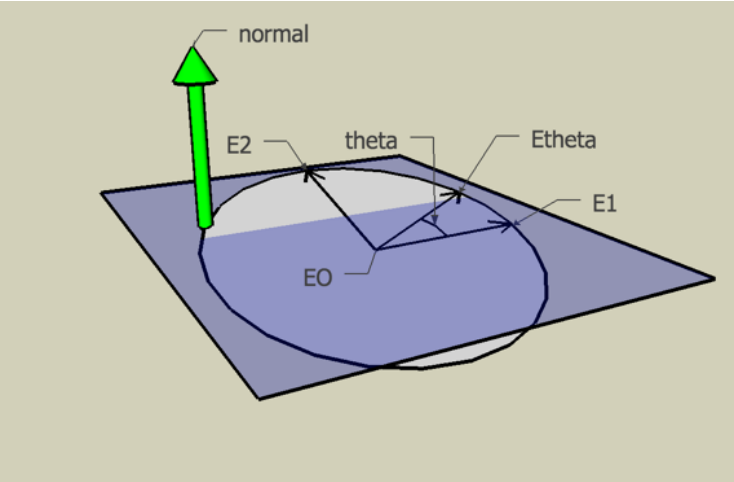


Figure 2.5: Description of the methods to find the deepest point on one flat part of the dRoombotShape, and variables definitions.

We can express the depth $D(E_\theta)$ of a point E_θ on the circle, knowing the depth of the center of the circle (see figure 2.5 for geometric definition):

$$D(E_\theta) = D(E_0) - (cos(\theta)\vec{e}_1 + sin(\theta)\vec{e}_2)\,.\vec{n} \tag{2.1}$$

were θ, E_0, $\vec{e}_1$, $\vec{e}_2$ and E_θ are parameters defined in the figure 2.5. These parameters are precomputed values for each of the flat part of the shape. $\vec{n}$ is the normal of the plane we test for collision.

by deriving the expression 2.1 among θ, we found the expression of θ_{min} and θ_{max} :

$$\begin{cases} \theta_{min} = arctan(\dfrac{\vec{n}.\vec{e}_2}{\vec{n}.\vec{e1}}) \quad ,\theta_{max} = \theta_{min} + \dfrac{\pi}{2} \\ or \\ \theta_{max} = arctan(\dfrac{\vec{n}.\vec{e}_2}{\vec{n}.\vec{e1}}) \quad ,\theta_{min} = \theta_{max} + \dfrac{\pi}{2} \end{cases} \tag{2.2}$$

Then if both $D(E_{\theta_{min}})$ and $D(E_{\theta_{max}})$ are positive, the ROOMBOTS shape must lie on the whole surface. if only $D(E_{\theta_{max}})$ is positive, it means that we have only one contact point, $E_{\theta_{max}}$.

In all cases, the normals returned for the contact points are the normal of the plane we test.

2.2.3 Implementation and result.

The harder work on this part was to make Webots able to use this new dGeomClass in simulation. We use an physics plugins in order to replace the dGeomSphere by our new dRoombotShape. As webots was not fully designed to do this, there are still some bugs that cannot be easily corrected without having access to the main Webots source code. But we still succeed launching a simulation of one ROOMBOTS module, standing vertically on the floor.

But the simulation is very unstable. Indeed, when we want the module to lie horizontally, it cannot succeed staying on two flat surface, and undamped oscillations appears. This is not surprising since unilateral physics constraint resolution (in our case contact detection) is an hard problem in simulation [ISI].

So, for further experiment, we cannot use this, until we correct this computation instability. In the other experiment described in this report, this work was not used.

2.3 libOptimizer : an object oriented optimization library.

Another part of our objective was to test several optimization algorithms. Then ss for the Central Pattern Generator, we created a Library that let us test several algorithms without having to recompile our main program, and by setting the parameter through the XML configuration file. For this libray, we make an original choice of design in order to help us parallelize the simulation when the optimization algorithm permit it. Indeed, most of the optimization libraries like COOOL [DG] and LIBGA [lib] use trackback function to compute the Objective function of the algorithm. Here we present another design, that is particularly helpful to repartize the computation among several computer for distribuable optimization algorithms, like Particule Swarm Optimization (PSO) and Genetic algorithms (GA).

2.3.1 General organization.

The library lies on two main classes Optimizer and OFCalculator:

- Optimizer : This class contains the main parameter of the choosen optimization algorithm, and runs its inner computation.
- OFCalculator (standing for Objective Function Calculator) : who is just an abstract class, who must be implemented by the user to adress his problem, and that must provide the computation of the objective function of the problem.

The two classes work together like this :

1. Optimizer runs some internal optimization computation, and then asks OFCalculator to estimate one or several time the objective function, then it give the hand to OFCalculator.
2. OFCalculator have to compute the Estimation. Here the user can parallelize the computation if the Optimizer asks for several estimation at one time. Then it sends back the value of the estimation to the Optimizer.

3. Each time the Optimizer get back an estimation of te objective functiun, he looks if he need to do more optimization. If not it stops, else we go back to step 2.

As for the libCPG, a factory has been provided for creating and setting up the parameter from a configuration file.

At the moment, only PSO has been programmed in this library. This library has been used to make a program OptimizerServer, that just run the optimization for a CPG defined by an XML configuration file, on paralize the code between several computers.

2.4 worldMaker, a tool for fast ROOMBOTS design.

The last tool which was made for this project, in collaboration with Mikaël Mayer, has been worldMaker. This program takes the XML configuration file that defines an assembly of ROOMBOTS modules as input, and then give back the corresponding .wbt file for running in Webots. This tools helps the user to rapidly design a ROOMBOTS assembly, avoiding the user to manually compute and enter the translation and rotation fields in Webots to positionate the modules.

Chapter 3

Study of a specific case : the quadruped ROOMBOTS.

Before to work on undetermined ROOMBOTS robots, a specific robot has been created and carefully studied. The goal, was to get more skilled about manipulating Roombots, but also try to understand how we can generate displacement. In chapter 4 we tried to put the result of this study in an automated way to design Central Pattern Generator. We put on the section above, the way of thinking we use to design such a robot

3.1 Experimental set up.

3.1.1 Structural description of the robot.

We choose to first design a quadruped like robot. Indeed quadruped locomotion has been well studied for a long time ([Hil65]), works has been done at BIRG on generic CPG that cover most of the possible quadruped gait [RI08], and this robot was also used in former project [Lé08], letting us compare results. The robot is made of five ROOMBOTS modules, and organized as seen in figure 3.1.

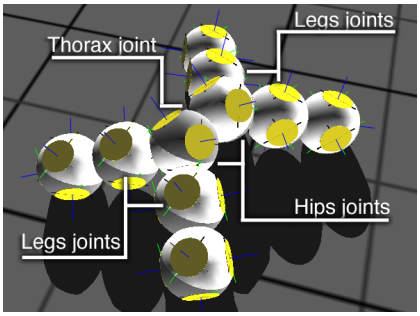


Figure 3.1: Representation in Webots of our quadruped robot, and joint nomenclature.

The main idea when designing this robot, was to have Legs able to make a symmetric motion. As descibed in [Wie08], this will help us to reduce significantly the search space of the optimization algorithm, as each Neuron driving one leg actuators will share its parameters with the neurons at corresponding legs.

But when as we get a symmetrical motion for the leg, if we do nothing else, most of the time, the 4 leg will touch the floor. Then the effect of one leg getting backward in order to push the robot forward, will be reduce by the movement

of other leg getting forward to prepare for next step. We need to reduce the leg friction with ground. Therefore we want the robots to be able to select which leg will support its weight. Then we positionned the central module, such a way he was able to do this by acting on his "Thorax joint" (see figure 3.1).

3.1.2 Design of the Central Pattern Generator.

The next step was to imagine a Central Pattern Generator that can make this structure able to move. Way of locomotion can be found, as a car style locomotion [Lé08], or more bio-inspired gait. We choose the first one has the more scalable, since it depends less on the structure (for the first one, legs have to be able to align their "wheels"), and has ROOMBOTS servo cannot exceed a low velocity 25 turn per minute, leading to maximal velocity (according the diameter of the wheel) of $2\pi.0,061.\frac{25}{60} = 0,160m.s^{-1}$.

Then we choose to use a amplitude-controlled phase oscillator network as a Central Pattern Generator for our robots. This type of CPG has been choosen as one of the simplest one [Ijs08b]. Indeed, we can directly specify the phase difference between the neuron, on the contrary of Hopf oscillators, where the difference phase depend from both the Amplitude and the coupling weight of the oscillators.

This network is parametrized in two ways : the parameter of each neuron, and the topology and value of coupling (see section 1.2.1). For each neuron, we cant set their amplitude,frequency and offset. Determining the topology of the network, lead to choose whether two neurons are connected or not, leading to $N(N-1)$ discrete parameter to determine (where N is the total number of neuron), and each link is defined by two continous parameter (the weight and the phase bias).

In our case, it will lead to a huge amount of parameters, and that is not feasible to train such a network if we don't fix some of the parameter, as we can't exceed 10 parameter if we don't want our optimization to exceed a few hours.

Reducing the number of parameters.

We tried to find a good trade off between the number of open parameters of the network (parameter to optimize), and not limiting too much the movement of the robot. Our idea is to give a guess of one solution, and optimize some parameters of this solution.

First we don't put neuron that drive the two low end joint of each leg segment. Indeed, the movement of this joint has only insignificant effect on the motion of the structure, so we just drop this neuron.

Then we choose our gait to be a symetrical trot. Indeed, our robot is only able to choose on which pair of diametrically opposed leg it wants to stand, when acting on his hip joint. This is typically the behavior of a trot motion, as diametrically opposed leg are in phase in their moves. This choice lead to completely determine the network among leg joint neurons.

Then we choose tho implement the network described in figure 3.2. In this network, we choose to fix some of the parameters :

- the phase bias between legs and hip joints. its value was choosen in order to

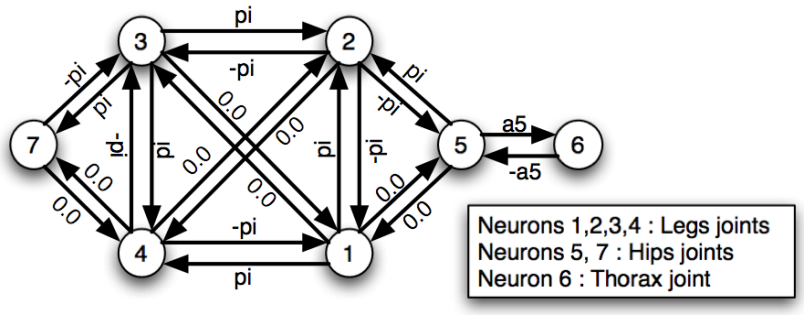


Figure 3.2: Topology of the coupled oscillators network we use as CPG. All the weight of the link are fixed to one, and the biais used figure on the graph edges. a_5 is one open parameter for the optimization.

act synchronousliy with leg joint neuron, to have a maximal displacement of the leg.

- the offset of hip and thorax joint, in order to have a symmetric structure when all output is zero.
- the phase bias between leg joints, in order to systematically have a symmetric trot.

Another parameters will be determined by the optimization :

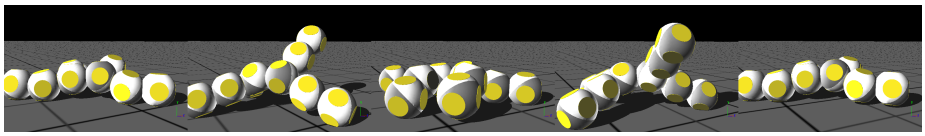
- The amplitude and the offset of the leg joints. Since all legs are symmetric, this value is the same for the 4 neurons.
- The amplitude of hip and thorax joints. As hip are symmetrical, we just use the same value.
- the phase bias between hips and thorax joints.

Another problem we discovered of the design, is how the neuron must act on servomotors. By convention, we want for example, that when the value of a neuron is increasing, the movement of his leg. To do this, we have to "reverse" the output of the leg neurons of one side ,i.e. make the servo of one side to follow the opposite of the value of the neuron . This help us to clearly separate the synchronisation of the network, and how he act on the structure.

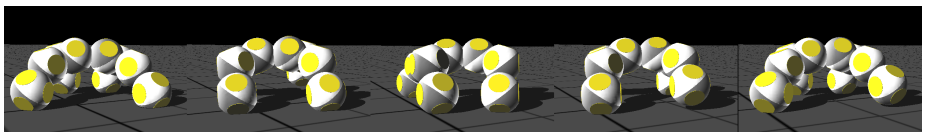
Optimization processus.

In order to train the network, we have to use an optimization algorithm, that tries to maximize the distance made by the robot in a certain amount of time. As proposed by [Wie08] [May07] [Mar05], we have to keep this simulation high (30 seconde in our case) to avoid the robot take an advantage of his strating position (by first make a big jump and falling down).

In order to have a smooth objective function, we change slightly the problem and make the optimizer minimize the value $\frac{1}{1+D}$ where D is the covered distance. This help to smooth the objective function, and keep it the $[0; 1]$ interval.



(a) with frequency fixed at $0,25Hz$



(b) with frequency found by optimization ($0,8113Hz$)

Figure 3.3: The two optimized gait.

Another key result of the former project, on both ROOMBOTS and YAMOR locomotion, is the fact that this objective function is multi-modal. Then the use of the Powell's method like in [May07] and [Wie08] often ends in a local minima. Then we choose to use an Particule Swarm Optimization [PSO], an evolutionnary algorithm, that is more robust not falling in local minima.

3.2 Results.

3.2.1 Optimization of the frequency.

We ran the optimization twice, first by setting the frequency value of the network at an arbitrary value of $0,25Hz$, and a second time, by letting the frequency be optimized. Surprisingly the two motions differs a lot (see figure 3.3). This could be explain as, the global speed doesn't depends linearly from the speed, as proposed by [Wie08]. The best motion at higher frequency is not just the acceleration of the best low-frequency motion, there are lot of dynamical (in a physical sense) effects we need to take in account. This assumption is also verified by the fact that the optimal frequency found in the second experiment, is not the maximal frequency : even if the optimizer was told not to go further than $1Hz$, the optimal gait that was found has frequency of $0,8113Hz$.

3.2.2 Control of the servomotors.

This two experiments point out a new problem in the control of the servomotors. Indeed, at high frequency, the servo failed to follow the motion ordered by his neuron, as seen in figure 3.4. This could be explain as a physical limitation of the ROOMBOTS servomotors. Their speed cannot exceed 25 turn per minute ($2,618rad.s^{-1}$) which is a quite low. Then we need not to ask for motion that will overflow this value like in figure 3.4 (you can see the difference between the slope of the servo curve, and the maximal slope required by the neuron). Another reason is because Webots use a simple P-conntroller for driving the servo. The use of a PID-controller should gave much more results.

One can remark that we could just count for the optimization process to handle this problem. Sure the optimization process will learn the default of the servo controller. But here, we are in simulation, but in real case, this lack on

consistancy in the control, lead to tremendous effect, as we not correctly drive the servo, we are not sure that the gait will periodically reproduce. We will fail in controlling the displacement of the robot.

One first solution is to limit the amplitude by adding a constraint in the optimization algorithm. In our case, the maximal speed a neuron will is given by the expression $2\pi.f.A$ were A is the amplitude of the neuron and f his frequency. But we think it is not the best solution, since :

- it is not scalable to other CPG, like the Righetti's one [RI08], as there is no simple expresion of this maximal speed, especially when using a duty ratio different of 1.

- even if the command of the servo , don't ask to overflow this max speed, there is cases, as in figure 3.5, were some neuron fail to reach the neuron trajectory. Further more, servos that have the same trajectory to follow (as neuron 5 and 7 in figure 3.5), could both succeed or failed to track their neuron trajectory, depending on their place in the neuron structure. Even if we use PID-controller, they will need to be tuned, depending on the position of the servos in the structure, and the whole motion of the robots.

Then it appears that one correct slution to adress this problem, is to add some control feedback in the CPG, to, at least, have servos and neurons trajectories synchronized.

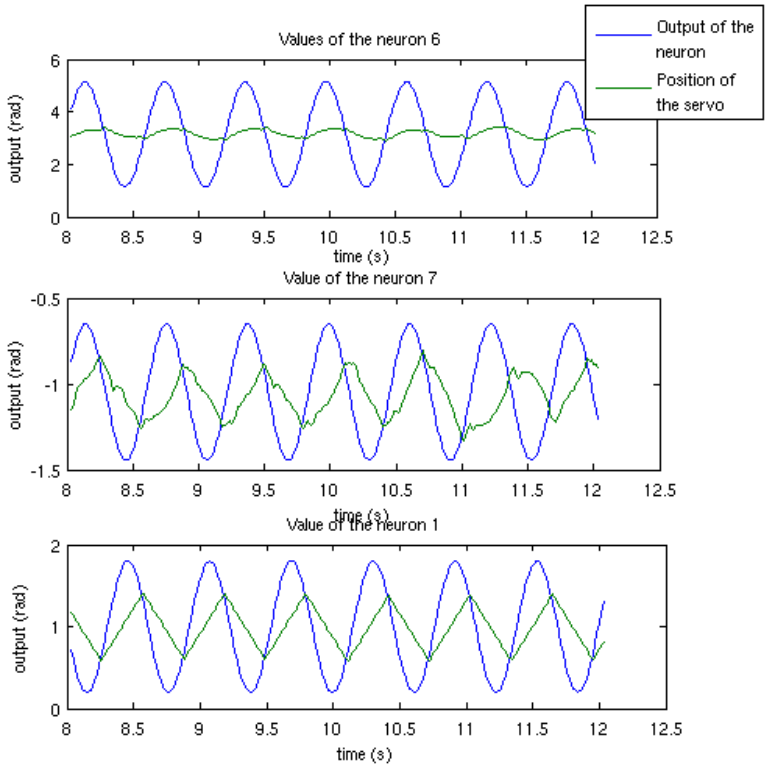


Figure 3.4: Servos error in tracking neurons output at high frequency$(0,8113Hz)$.

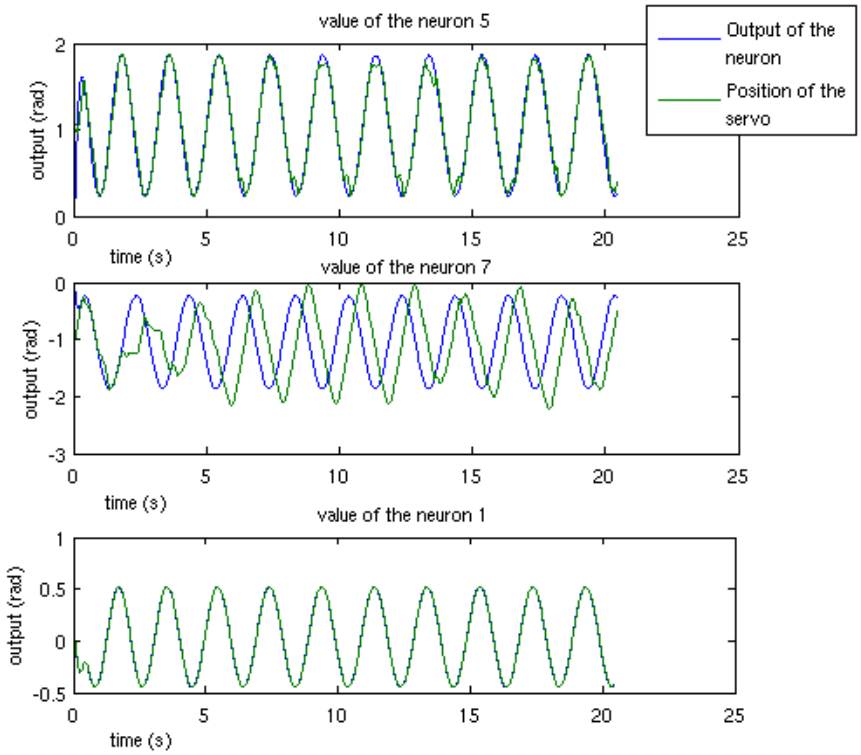


Figure 3.5: Servos error in tracking neurons output at low frequency $(0,25Hz$

Chapter 4

Towards an automatical way to design Central Pattern Generators.

In this chapter, we will discuss a way of automatically design a Central Pattern Generator for a CPG. Our a problem is, given any ROOMBOTS robot with any morphologicval structure, how can we best design a CPG that will lead to a good locomotion controller for this specific structure ? It is sure that there will be structure better suited for locomotion than other, and the quality of the gait is highly determined by the quality of the structure, but we want to automatize the design process for the CPG. Even if this topic was one of our main goal, due to a lack of time, the solution proposed hasn't been tested yet. Therefore we just expose our reflexion and justify some ideas, that we hope, will lead to such a behavior in modular robotic.

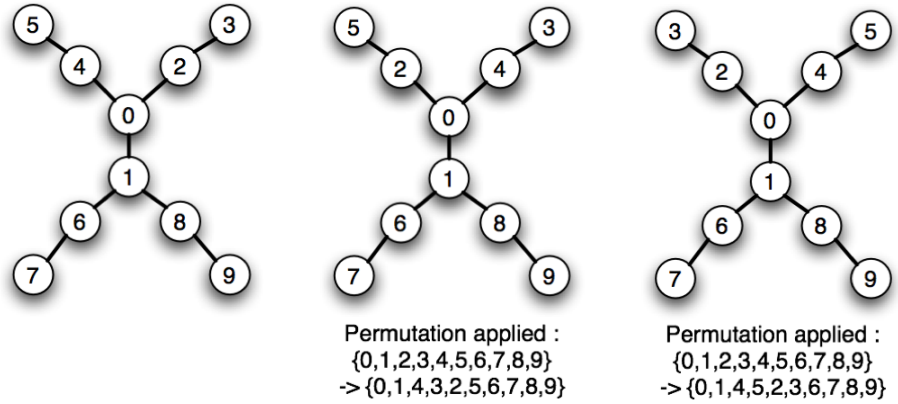
4.1 Context.

First and foremost, we want to remind some graph theory results. Indeed, since we study chain type robot, we can fully characterize the way the modules are assembled each other by a graph. In this graph each vertex represents a module, and each edge represents a mechanical connection, and owns a label that described how this connection was made (see [Lot09] for more details). Then this representation could be useful in order to find morphological symmetries. In further section, we will name the graph the structural graph or the hierarchical graph.

4.1.1 Symmetries in graph theory.

The following line are part of the bliss package documentation [JK08], a tool for finding symmetries in graph

A coloured graph $G(V, E, c)$ is defined by a set of vertice indexed by integers $V = \{1, ..., n\}$, a set of edges,$E \subset \{(v_1, v_2) \in V \times V\}$, and a colour map $c : V \rightarrow \mathbb{N}^+$, a function that to each vertex associate a color as a postive integer.



(a) Original graph (b) An non automorphism permutation (c) An automorphism

Figure 4.1: Example of an automorphism and a non-automorphism permutation of a given graph.

Let $\gamma : V \to V$ be a permutation of V. We defined the *permuted coloured graph* $\gamma(G) = G\left(\gamma(V), \{(\gamma(v_1), \gamma(v2))|(v_1, v_2) \in E\}, c^\gamma\right)$ where $\begin{array}{lll} c^\gamma : V & \to \mathbb{N} \\ v & \mapsto c(\gamma^{-1}(v)) \end{array}$. the permuted graph is then obtained by permuting vertice and edges according to gamma, but vertice keep their color.

The equivalent of a symmetry for a graph is called an *automorphism*, a permutation γ as $\gamma(G) = G$ (see figure 4.1).

One important result is, that the set of automorphisms of a given graph is an *finite group* for the composition operator, i.e., given two automorphism γ_1 and $gamma_2$, $\gamma_1 \circ \gamma_2$ is also an automorphism.

As a finite group, the set of automorphism $auto(G)$ of an graph, owns a set of *automorphism generarators* $gen(G)$, following this proporty :

$$\forall \gamma \in auto(G), \exists n \in \mathbb{N} \,/\, \exists(\gamma_1, \gamma_2, ..., \gamma_n) \in gen(G) \,/\, \gamma = \gamma_1 \circ \gamma_2 \circ ... \circ \gamma_n \quad (4.1)$$

Then the set of generator of G, contains all the information about symmetries in the graph. The main purpose of the bliss package, is to determine the set of automorphism generator of a given graph, and an approximation of the total number of isomorphism in graph. The generators can also be seen as the smallest automorphisms of G.

Another interesting general result of graph theory, is that all the assymetrical graphs (i.e. the first graph which only have the identity map as automorphism), have more than six vertices. Therefore, small graph, as the ones we manipulate with ROOMBOTS , are more likelly to have lot of symmetries.

4.1.2 Expression of our problem.

As we expose in chapter 3 to fully parametrize a Central Pattern Generator lead to a huge optimization problem in term of dimensionnality.

We need to :

- set up the topology of the oscillator network. This give us $N(N-1)$ discrete parameters to otpimize.

- set up all the continous parameters of the network.

For example, for an network of four neurons (the minimum for a quadruped), each neuron requiring three parameters, and each link one, we need to run $2^{4.3} = 4096$ continous optimization algorithm, each one having between 15 and 35 parameters. As we want to perform online optimization, such a naïve solution is not approachable.

Therefore an idea exposed by [Wie08] is to use morphological symmetries in order to reduce this number, according to some rules exposed in section 4.1.2. First, our main goal was therefore to define an algorithm that will look for such symmetries. Here, according to some observation we made when studying the quadruped robot case, we propose to go a little further, and make more choice to decrease the number of parameter even more.

About symmetries in coupled oscillator network.

The principle exposed by [Wie08] , are founded by the H/K theorem, as explained in [RI08], and more detailled in [GS06]. The main idea is, that if the network has some topological symmetries in the sense of graph theory, then the same spatio-temporal symmetries can be found in the resulting output of the network. Then [RI08], use this property backward, in order to determine, for a desired gait, which are the required topology of the network.

This bring us an important result : we don't need to test all the possible topologies for a network. If our algorithm is able to classify, whether our structure is a tripod, a quadruped or a hexapod, then we can reduce our search by testing among a small amount of type of locomotion. For example for a quadruped, we can test if the structure is better suited for a trotting gait or a bounce gait, by training a CPG for both gait, and choose the best one.

Implicit choices we made when designing our quadruped CPG.

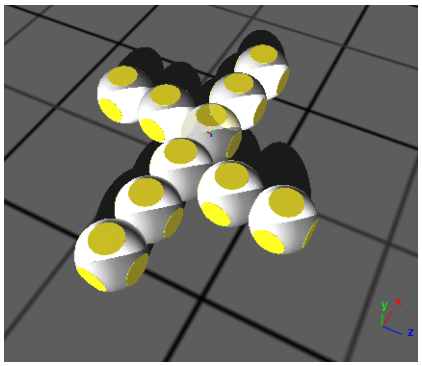
First we want to come back to the study of the quadruped ROOMBOTS . Our way of thinking was to make the most symmetrical structure possible, and then let the CPG play with this symmetry in order in the most profitable way. In order to maximize the number of symetries in the structure, we choose some specific value for the Hips and Thorax servos respectively $\pm 60°$ and $180°$. This leads to two big conclusion :

- Hierarchical symmetry of two modules (i.e. the fact that two modules are symmetrics in the representation graph of our structure) is only a *prerequisite* for morphological symmetry (spatial symmetry). Some positions of the servo must be fixed, in order to ensure spacial symmetries.

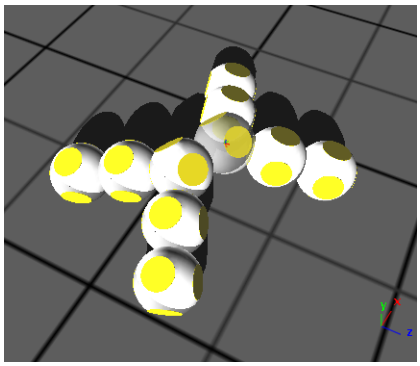
- We must classify the joints between two typs :

  - these which must have their offset fixed, making the structure to work around a symmetrical posture.

  - these which have their offset totally free.



(a) Structure with only hierarchical symmetries.



(b) Structure with morphological symmetries.

Figure 4.2: Distinction between morphological and hierarchical symmetries in a ROOMBOTS structure(the previous quadruped robot) : the two robots only differs by the position of the "body" servomotors (the previous hips and thorax joints, see section 3.1.1.

This second conclusion is linked with another remark we can make when we designed our CPG. We implicitly make a distinction between modules : we choose these which will be use as effector, and these which will be part of the body.

Then we want our algorithm to be able to make the distinction between effector and body part, and then find the value of the body servo motor that leads to the more symmetrical structure.

4.2 Presentation of our solution.

We propose an algorithm that can lead to an automatic design of the CPG. This algorithm act on four step :

1. Classify the modules, between those which are body parts, and those which are effectors.
2. determine the hierarchical symmetries of the structure, by obtaining the hierarchical graph automorphism generators.
3. Find the best symmetrical structures we could obtain by acting on the body part position. The "symmetricality" of the structure is determined with the help of the hierarchical symmetry we found.
4. Set up a Central Pattern Generator, according to some heuristic rules.

In order to simplify our problem, we make an assumption : we assume that our hierarchical graph has no cycle. This is not a so strong assumption, since if the ROOMBOTS modules are assembled in a closed loop, there will be a lot of kinematics contraints, that will make our problem much more harder, even if we try to design a CPG model by hand. Then if the graph has no cycle, we could see it as a tree.

4.2.1 Modules classification.

A classification of the module between body part and effector could be easily made from the structural graph. Indeed we choose to classify whether a module is an effector according the two following rules :

- A module that has no children in the graph is an "effector".
- A module that has only one child, and, if this child is an "effector", then, then this module is an "effector".

All other modules are classified as "body" module.

Algorithm 1 Post fixed, depth-first search for handling module classification.

```
function explore_node( node N )
if N.children is empty then
  Mark N as "effector"
  return
end if
for M ∈ N.children do
  explore_node(M)
end for
if size(N.children) is 1 then
  if N.firstChild is marked as "effector" then
    Mark N as "effector"
  else
    Mark N as "body"
  end if
else
  Mark N as "body"
end if
end function

explore_node(root_node)
```

Such a classification, could easily be made by a post-fixed depth first search in the tree (see algorithm 1). But such an algorithm is not robust to degenerated cases, while for example, an effector module is taken for the root of the tree. Then prior to do the depth first search, we must elect a root module, such as the tree has a minimal depth.

4.2.2 Finding the most symmetrical structure as an optimization result.

This step is the main step of the algorithm. the main idea is to run an optimization algorithm, that will act on the servo position of the "body" module, in order to obtain the best symmetrical structure. The problem, as for all optimization algorithm, is to find a way to characterize how much a solution is symmetric, i.e. how we compute our objective function.

Determining the objective function.

Here we can be helped in this computation by the determination of the automorphism generator of the hierarchical map :

- to be hierarchically symmetric is a pre-requisite for being morphologically symmetric. Then our goal is to see if the hierarchical symmetries found can be transformed in morphological symmetries.

- the set of generators is the smallest subset of the automorphism group that "summarize" the hierarchically symmetries. Then we only have to test symmetries among the one given by the generators.

- the generators are the "smallest" elements of the set of automorphisms of the hierarchical graph. Therefore, we remark, empirically, that in most of the cases, the permutation will only swaps a small part of the modules.

Then the generator can provide us, pairs of module, we want to test their symmetry. And thanks to the kinematics model provided in [May09], it is a problem we can handle. If we provide all the servo positions, this model let us compute the homogenous transformation matrix that goes from one module referential to the other. Then a well-know result of linear algebra gives us a condition on the rotational part of this matrix $R \in \mathbb{R}_{3\times 2}$, for the two modules to be symmetric :

$$R^2 = Id \tag{4.2}$$

Then we can qualify how much two module are close to be symmetric, simply by computing the distance of R^2 to the identity matrix. As for the rigid-motion problem (see [Kum96]), the expression of this distance could be more consistant, when computed in a particular space. For example, one can use of the exponential map, as proposed by [Kum96].

Then the expression of our objective function is the sum of all this distances, for all potential pair of module found by our symmetries finder.

Summary of the main optimization algorithm.

Then for finding the most symmetrical structure, we can act like this :

1. set all the position of the servo of the "effectors" modules to be at zero.

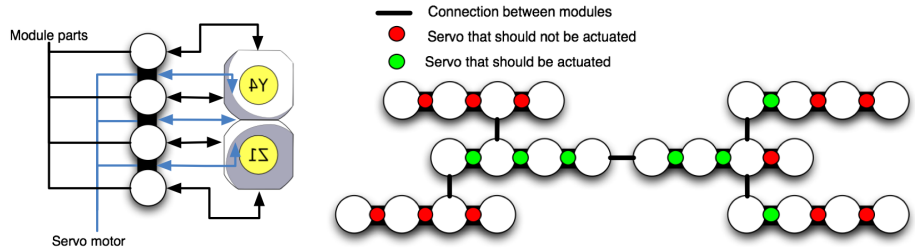
2. optimize the position of all the servos of "body" module, in order to obtain the most symmetrical structure, in the sense of the objective function described higher.

This step could provide us, as well as the best the solution, the pairs of modules that actually are morphologically symmetric.

4.2.3 Setting up the whole Central Pattern Generators.

In this section we describe some heuristic rules for setting the whole CPG. There could be three steps when performing this last part of the main algorithm :

- Determine which joints must be driven by a neuron.



(a) Correspondance between schematic and module

(b) Summary

Figure 4.3: Summary of the heuristic rules for determining which joints should be actuated or not.

- Setting up the topology for the "effector" neurons.
- Setting up the topology for the "body" neurons.

Determining which joints should be actuated.

This task is more doing some cleaning, by avoiding to drive a joint that will not act on the global motion of the structure. There are two rules for determining these joints :

- The joints that are flanked by two other modules should be actuated.
- For end-effector module, i.e. module that have no child in the hierarchical graph, only extremal servo could be actuated, and only if the corresponding extremal part is connected to the parent.

These two rules are illustrated in the figure 4.3.

Determining the topology for the "effector" neurons.

For these neurons, there are two things that must be done :

- Determine the topolgy between neurons on the same leg. We choose to simply follow the topology of the hierarchical graph, in order to put neurons in series, the neuron that is nearest to body part, is taken as the master neuron of the leg. Then the other oscillators, synchronize to its phase. We left open for optimization all parameters, for both link and amplitude.
- Determine the topology between legs. This could be done by just setting a precomputed sub graph for master neurons defined higher. For example, when we are facing a quadruped robot, and using Hopf oscillator as CPG model, we can take on of the coupling matrix proposed by [RI08].

Then we must only predefine by hand, some network topology for each number of legs possible. For this purpose, we can directly take inspiration from nature, as there are example for lot of possible configuration in, snakes (for one-leged robots), mammals (most of them quadruped), insects (hexapode), spiders

(octopode), and for a huge number of legs, one can take inspiration on the centipede, were some work about the modelisation of this locomotion is currently done at BIRG [Ijs08b].

We can remark, that we can reduce the number of open parameters of this part of the network, by taking the same value for neurons and links parameters of legs, that actually share a morphological symmetries. Such pairs were found at the previous step of the main algorithm (see section 4.2.2).

Setting the topology for the "body" neurons.

Here we don't find any clever way than to reduce the amounts of parameters by making the topology as simple as possible, by putting the oscillators in a chain. the this chain is connected to one of the legs master neuron. We can notice that for CPG models based on phase oscillators, the addition of this part will not disturb the main synchronization of the legs, but for other oscillators type as Hopf oscillators, further work must be done, to see if it is not the case. This work may be done, by using the result of the H/K theorem [RI08], [GS06]. But as the concept of the groupoïd are tough, this he a work we were not able to do.

Remark :We must recall that the offset of this neurons ouput is fixed, since we want the structure to operate around the most symmetrical case. Then the offset of "body" neurons are fixed to the value found in the optimization processus.

4.3 Necessary Implementation steps.

Due to a lack of time, we were not able to start the implementation of this ideas. So, again, we must notice, that it is just theorical reflexion, that need to be better mathematically formed. But we wanted to provide our reflexion in order to help further work on the subject. We can also notice that there is still dark points about the processus, we must resolve :

- how could we, from when we got the automorphism generator, determine the couple of module we have to test for morphological symmetry. Indeed, as seen in the figure 4.1, there could be more than two module that are involved in a generator. Then some work has to be done to handle these cases.

- Again, we must find a way to graft the "body" part of the network to the "effector" one, without disturbing the synchronization we have predefined for the "effector" part of the network.

- some points we forgot, since we haven't really try this solution, even if we try to formalize it as well as possible.

Chapter 5

Conclusion

This project, as a further step of former BIRG projects on modular robotics locomotion, tried to find out some solutions to problem that appear when passing from the YaMoR modules to the Roombots one. After having presented some software tools we made to manipulate Central Pattern Generator, Optimization Algorithm, and improve the quality of Roombots simulation, we exposed how we have design a CPG models for a specific Roombots robots. Then we proposed an algorithm that automatically find symmetries in a Roombots structures. This step was a pre-requisite for setting heuristic rules that let us automatically determine a CPG for a given Roombots robot, and let us reducing the optimization search space, which is a pre-requisite for online learning.

The study of the quadruped Roombots motion, have let us observe a limitation of their servomotors as their speed cannot exceed a given value. We remark that we can solve this problem by both putting some constraints in the research space, and add control feedback between servomotors and CPG neurons. We also make a distinction between hierarchical and morphological symmetries. This last idea was presented has one of the main concept for the symmetries finding algorithm, that can be seen as an optimization algorithm that take as input parameters, the position of the servomotor of parts of roombots module. These modules are determine by classifying which module will be use as leg parts, and other as body parts, and we provide an algorithm to handle this problem. The the optimization algorithm is told to find the most symmetrical structure in a spatial sense.

Due to a lack of time, this algorithm is only a reflexion, that must be better mathematically formalized, and tested, in order to see if, indeed, he is a good solution that can lead to an automatization of the CPG design process.

Aknowledgment

Special thanks to ... My supervisor, Alexander Sproëwitz, who have been continously supporting me, even in the last minute, and for introducing me to this captivating subject.

Alessandro Crespi, for its kindness and availability, specially on the informatics field, where his advices has been a great help to me.

Mikaël Mayer, who give me good advices for develloping stuff, and for ever trying to made our common work better.

Jocelyne Lotfi, for both her ability to listen my complaints, and his patience, especially when me and Mikaël were in disagremment on code stuff.

Professor Auke Jan Ijspeert, who supervised me at high level, and for the quality of the lecture he gave in the "Models of Biologicaly Sensory-Motor Systems" class.

Yvan Bourquin, who has been a great help, specially when I was needed to go into the technical parts of Webots.

Appendix A

Syntax of the XML Roombots configuration file.

All the programs created in this project can be configured by one central configuration file. This file, contains all morphological informations about :

- the ROOMBOTS assembly structure,
- the Central Pattern Generator that drive the robot,
- the Optimizer we want to use.

The language retained for this file is XML, since it is a well-diffused standart, and their exist a lot of libraries that allow us to create/parse this file. This Appendix details the structure of this file. There is also a DTD file, defining the syntax we used in this document, and let parser automatically verify whether the syntax of the parsed file is comformed or not.

The root node of the XML document is called "<roombot_config>" and he can have three children (see figure A.1):

- "<module_tree>" a node containing all the hierarchical information, and Central Pattern Generator definition.
- "<parameter>" a node that contains some global information to provide to Webots.
- "<optimizer>" that contains information defining the optimization algorithm we should use.

A.1 Description of "<module_tree>" node

The <module_tree> node contains all the information of the hierarchical structure. The module description are hierarchized in a tree of <module> node. The root of the tree is directly a child of the <module_tree> node. One can point out that we must only to define the position of the root node, positions of the child are given by

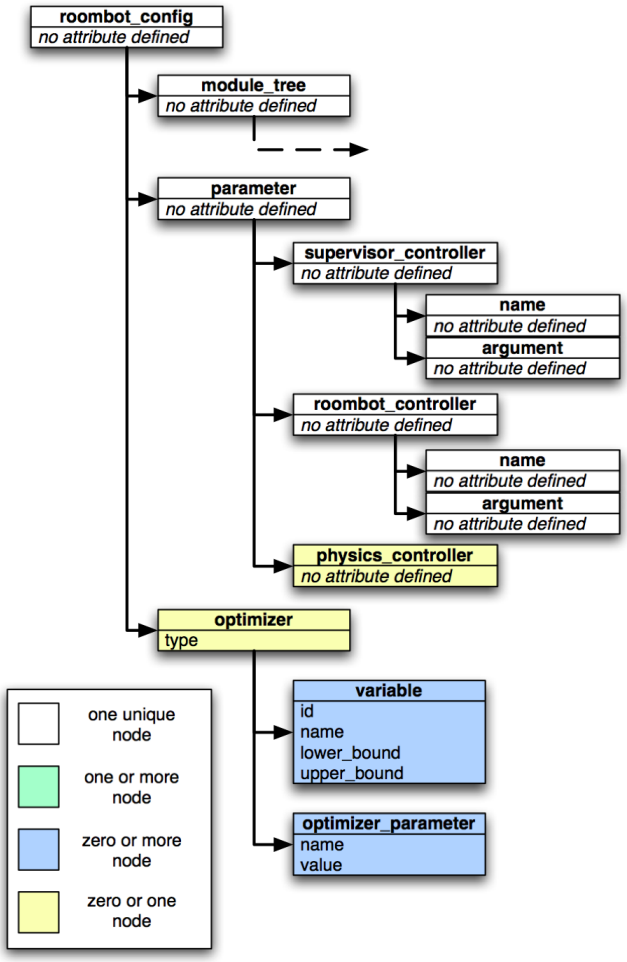


Figure A.1: Hierarchy of the XML file at a global scope (<module_tree> node sub hierarchy is missing, see figure A.2).

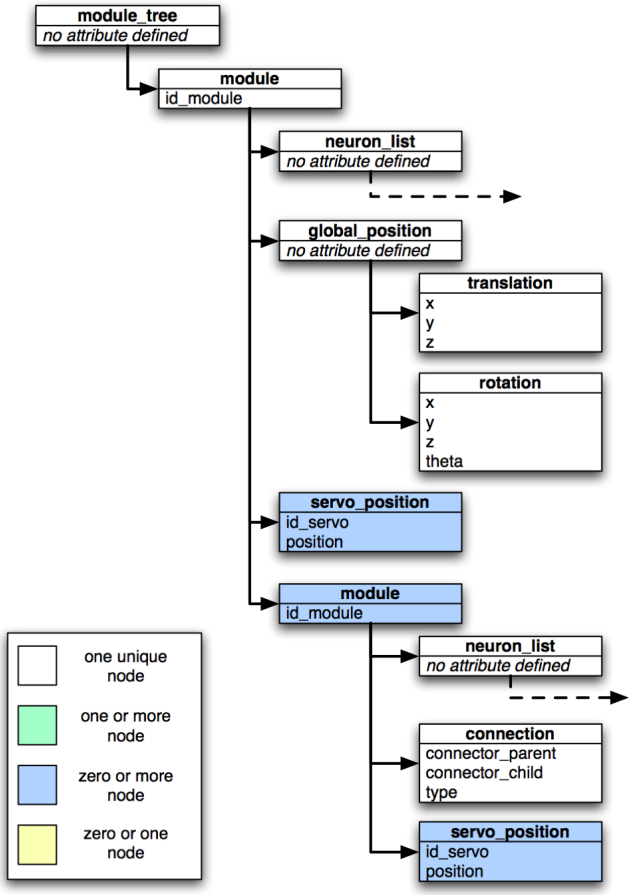


Figure A.2: Sub hierarchy of the <module_tree> (<neuron_list> node sub hierarchy is missing, see figure A.3).

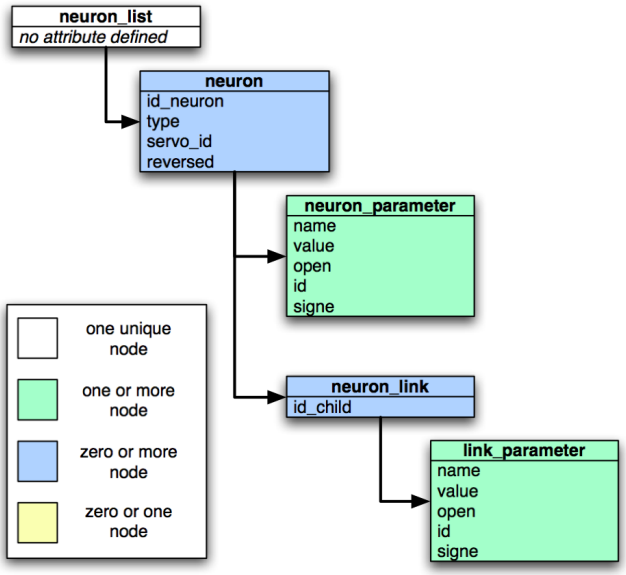


Figure A.3: Sub hierarchy of the <neuron_list>.

List of Figures

Bibliography

[ASB+08] Masoud Asadpour, Alexander Sproewitz, Aude Billard, Pierre Dillenbourg, and Auke Jan Ijspeert. Graph signature for self-reconfiguration planning. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems IROS 2008*, pages 863–869, 2008.

[Bou04] Yvan Bourquin. Self-organization of locomotion in modular robots. Technical report, Biologicaly Inspired Robotic Group, 2004.

[Cen97] Palto Alto Research Center. Polybot, a chain reconfiguration robot. `http://www2.parc.com/spl/projects/modrobots/chain/polybot/index.html`, 1997.

[CMLA07] Cyrille Collette, Alain Micaelli, Pierre Lemerle, and Claude Andriot. Dynamic balance control following disturbance of virtual humans. pages 734–744, 2007.

[DG] Lydia Deng and Wences Gouveia. Coool : The cwp object oriented optimization library. `http://coool.mines.edu/index.html.old`.

[FBHK91] Toshio Fukuda, Martin Buss, Hidemi Hosokai, and Yoshio Kawauchi. Cell structured robotic system CEBOT - control, planning and communication. *Robotics and Autonomous Systems*, 7:239–248, 1991.

[GS06] M. Golubitsky and I. Stewart. Nonlinear dynamics of networks: the groupoid formalism. *Bull. Amer. Math. Soc.*, 2006.

[Hil65] Milton Hildebrand. Symmetrical gaits of horses. *Science*, 150(3697):701–708, 1965.

[IC07] A.J. Ijspeert and A. Crespi. Online trajectory generation in an amphibious snake robot using a lamprey-like central pattern generator model. pages 262–268, 2007.

[Ijs08a] Auke Jan Ijspeert. Central pattern generators for locomotion control in animals and robots: a review. *Neural Networks*, 21(4):642–653, 2008.

[Ijs08b] Auke Jan Ijspeert. Models of biological sensory-motor systems. Lecture Notes, 2008.

[ISI] CEA-List & ISIR. Arboris un simulateur de chaîne de solide polyarticulées. `https://vizir.robot.jussieu.fr/trac/arboris/`.

[JK08] Tommi Junttila and Peter Kaski. *bliss*: A tool for computing automorphism groups and canonical labelings of graphs. `http://www.tcs.hut.fi/Software/bliss/index.html`, 2008. Laboratory of Theorical Computer Science, Helsinky University of Technology.

[KKK+03] S. Kajita, F. Kanehiro, K. Kaneko, K. Fujiwara, K. Harada, K. Yokoi, and H. Hirukawa. Biped walking pattern generation by using preview control of zero-moment point. In *Proc. IEEE International Conference on Robotics and Automation ICRA '03*, volume 2, pages 1620–1626 vol.2, 2003.

[Kum96] Zefran Vijay Kumar. Two methods for interpolating rigid body motions. In *IEEE Intern. Conf. on Robotics and Automation*, 1996.

[lib] Galib :a c++ library of genetic algorithm components. `http://lancet.mit.edu/galib-2.4/`.

[Lot09] Jocelyne Lotfi. Self-reconfiguration for adaptive furniture. Master's thesis, Ecole Polytechnique Fédérale de Lausanne, 2009.

[Lé08] Simon Lépine. Locomotion in Modular Robots: YaMoR Host 3 and Roombots. Semester project, Biologically Inspired Robotic Group, 2008.

[Mar05] Daniel Marbach. Evolution and online optimization of modular robot locomotion. Master's thesis, Ecole Polytechnique fédérale de Lausanne, 2005.

[May07] Jérôme Maye. Control of locomotion in modular robotics. Master's thesis, École Polytechnique Fédérale de Lausanne, 2007.

[May09] Mikaël Mayer. Roombot modules - kinematics considerations for moving optimizations. Semester project, Biologically Inspired Robotic Group, 2009.

[PCDG06] J. Pratt, J. Carff, S. Drakunov, and A. Goswami. Capture point: A step toward humanoid push recovery. In *Proc. 6th IEEE-RAS International Conference on Humanoid Robots*, pages 200–207, 2006.

[PSO] Particule swarm optimization: a tutorial. `http://www.swarmintelligence.org/tutorials.php`.

[RI08] L. Righetti and A. J. Ijspeert. Pattern generators with sensory feedback for the control of quadruped locomotion. In *Proc. IEEE International Conference on Robotics and Automation ICRA 2008*, pages 819–824, 19–23 May 2008.

[SABI08] A. Sproewitz, M. Asadpour, Y. Bourquin, and A.J. Ijspeert. An active connection mechanism for modular self-reconfigurable robotic systems based on physical latching. In *Proc. IEEE International Conference on Robotics and Automation ICRA 2008*, pages 3508–3513, 2008.

[ST08] Advenced Industrial Science and Technology. M-tran. http://unit.aist.go.jp/is/dsysd/mtran3/, August 2008.

[VKR08] Paulina Varshavskaya, Leslie Pack Kaelbling, and Daniela Rus. Automated design of adaptive controllers for modular robots using reinforcement learning. *Int. J. Rob. Res.*, 27(3-4):505–526, 2008.

[Wie06] P.-B. Wieber. Trajectory free linear model predictive control for stable walking in the presence of strong perturbations. In *Proc. 6th IEEE-RAS International Conference on Humanoid Robots*, pages 137–142, 2006.

[Wie08] Sandra Wieser. Locomotion in Modular Robotics : Roombot Module. Semester project, Biologically Inspired Robotic Group, 2008.

[Wik] Wikipedia. Self-reconfiguring modular robotics. http://en.wikipedia.org/wiki/Self-Reconfiguring_Modular_Robotics.

[Yer07] Michel Yerly. YaMoR Lifelong Learning. Master thesis, Ecole polytechnique Fédérale de Lausanne, 2007.

[ZCL07] Victor Zykov, Andrew Chan, and Hod Lipson. Molecubes: An open-source modular robotics kit. http://www.molecubes.org, November 2007.