Locomotion in Modular Robotics, Roombot modules

Sandra Wieser

June 6, 2008

Contents

Chapter 1

Introduction

Roombot is a new modular robot developed at BIRG, EPFL. It was initially developed for a "self reconfigurable pieces of furniture" project (See BIRG webpage), and changed its sharp many times until the current version. The Roombot Project deals with various fields of research. Other students have covered aspects like self-reconfiguration with Roombot modules, optimal shape of the modules, and locomotion of robots made of Roombots modules. This time the main novelties are: the introduction of passive elements in the robot structure, the implementation of a Central Pattern Generator for generating the command of the motors, and the possibility of use a motor in oscillation, but also in constant rotation.

Three robots will be designed and tested, and results are presented for two of these robots. These three robots presents three different locomotion gaits, that where found by three different strategies. The first three-legged robot is optimized with Powell's algorithm, has no coupling between the legs, and 7 open parameters. It was the first approach of locomotion with the Roombot modules. A four-legged robot is then tested. This time with only 2 open parameters, the results concentrates on the application of different gaits, and the influences of the two open parameters on the speed of the robot. Finally, a third three-legged robot is designed, to try to have better results and an increased stability.

This three different robots and strategies give a good introduction to the countless possibilities of modular robotics using Roombots. Chapter 4.2 gives an outline of these possibilities, and the three different strategies used for the three robots allow a complete discussion about the different parts of the project. The discussion chapter 6 gets back on the optimization method, and tries to determine the problems of the used methods and which alternative will be more pertinent.

Chapter 2

State of the Art

2.1 Modular Robotics

2.1.1 From nature to robotics

Modular Robotics is a huge domain of research. The global principle of modular robots is to conceive little robots with a very little number of degrees of freedom. Such robots are not very useful alone, but can combine one with each other to form a bigger robot that is more efficient. The key words for such robots are adaptability, reconfiguration and cooperation.

Figure 2.1: Image of a bridge made of ants: A single ant could not reach the other branch from the first one. By collaborating, they combine themselves with the other ants to form a bridge across which the colony can travel. *Formation d'un pont vivant chez la fourmi d'Argentine "Linepithema humile ".* CNRS Photothque - Guy Thraulaz

Inspiration for modular robotics comes from nature. We have some examples of

collaborative animals, which are more efficient in group than alone, like ants or gasp colonies. Ants can make bridges from their body to allow the colony passing across a canyon (Figure 2.1). They also collaborate for moving big charges like big insects cadavers, or falling from a tree in a drop of a lot of ants.

From here comes the idea of collaboration between different modules. The other fascinating property of living is the incredible ability of regeneration and adaptation that beings allow. The tail of a Salamander can entirely regenerate when it is cut, as the limbs of the sea star Sunflower (Figure 2.2). Animals have also an incredible potential of adaptability.

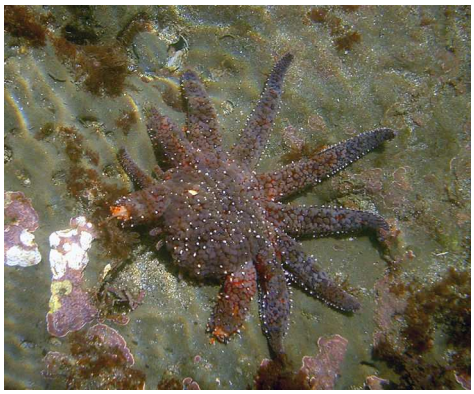


Figure 2.2: *Sea star is growing new legs after old ones were lost in Northern California.*Mila Zinkova, 2007. Such regeneration capabilities fascinate scientists, and one aim of modular self-reconfigurable robotics is to imitate this regeneration process.

Modular Robots have the aim to resolve this three biological problems by three different strategies. Collaboration/Adaptation by changing the robot configuration and architecture for adapting to the environment (Figure 2.3): adopting a snake-like structure for passing threw little passages between two walls, a spider-like structure for climbing on obstacles and a wheel-like structure for moving fast on flat grounds. Regeneration by self-reconfiguration(Figure 2.4): the broken part of the robot will be removed and another module will replace this lost limb. And adaptation again by long-life learning(Figure 2.5): the robot has to be aware of it's body or structure, detect when the current gait is not efficient any more and find a new gait of locomotion by its own.

2.1.2 Modular robots classification

The modular robots can be lattice type or chain type(Figure 2.6). A lattice structure is when the modules can fill a 2D or 3D space. Moreover the modules have to be able to move on the lattice structure. See image 2.7 for a 2D lattice example. A chain structure is when the modules are connected one to another in a string or tree topology. See image 2.8 for a chain-type robot.

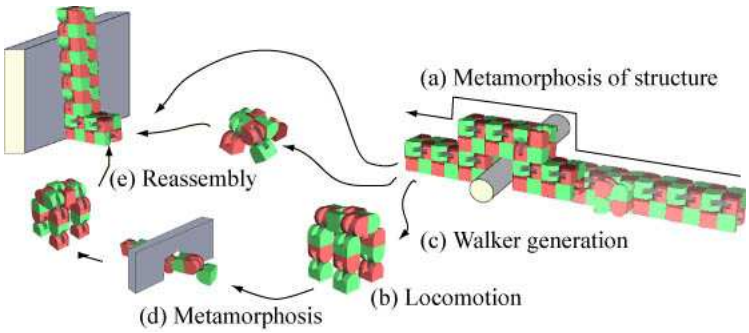


Figure 2.3: The potential of adaptability to new situations for the MTRAN modules. By reassembly, the robot can configure itself in different structures that allows climbing on obstacles, (a) and (e), locomotion (b), or taking reduced passages(d). This behaviors are inspired from the collaborative animals like the ants: little structures assemble and reassemble to make efficient robots that solve local problems. http://unit.aist.go.jp/is/dsysd/mtran3/what.htm

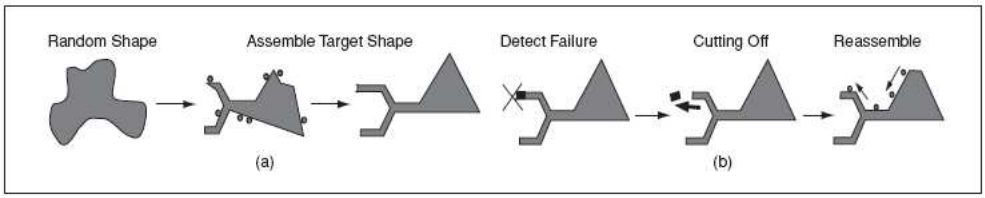


Figure 2.4: Schema of the concept of auto-assemble and regeneration in Modular Robotics. In (b), an unused part of the robot is carried to the missing part for replace it. [8]

Figure 2.5: Hob Lipson's spider robot is aware of its own structure and can find new locomotion gaits if the robot's structure changes (a broken limb for example). Such behaviors are researched in modular robotics because it allows to adapt to a new robot's structure.[3]

Whatever type the modular robot is, there are 2 different field of research on these modular robots. The first one is the self reconfiguration of the structure robot, the second one is robot locomotion.

Self-reconfigurations problems are easy to understand. The problem can be set as: Let H be a given structure of the robot. Which reconfiguration steps do we have to perform on the structure to reach a desired structure H' (Figure 2.9. Of course we would want a solution for any couple of structure (H;H'). We can also imagine that we search a solution for a unique H and given all possible H'. In this case, if

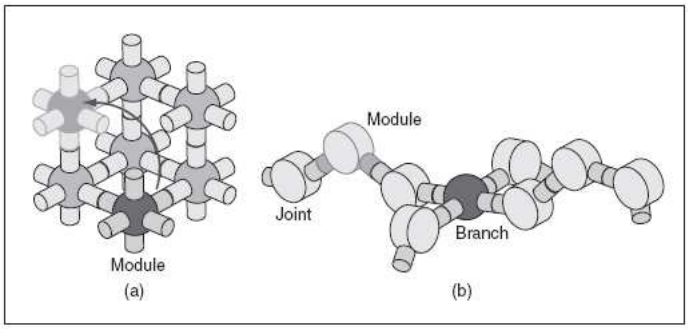
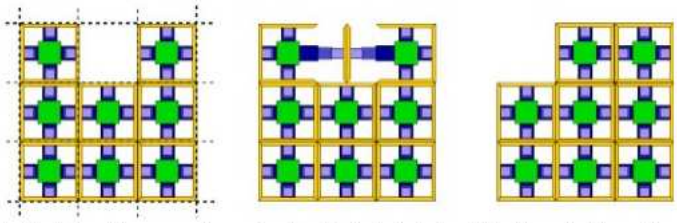


Figure 2. Types of SR robots. (a) Lattice type. (b) Chain type.

Figure 2.6: The two types of Self-Reconfigurable Robots, also called Modular Robots, schematically represented. a) shows a lattice type and b) a chain or tree type.[7]



In 2D with 8 modules an example reconfiguration is similar to the 9 piece sliding tile puzzle. The module in the upper left corner moves to the center.

Figure 2.7: Telecube modules in a 2D reconfigurable lattice. [10]

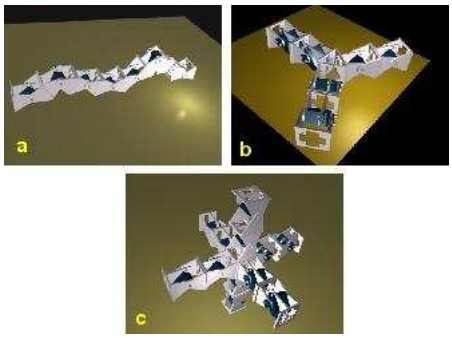


Figure 2.8: a chain type robot, a) with a string structure, b) with a tree structure. [5]

we want to go from a random structure H" to a structure H', we could do it in two steps: from H" to H, as we know its solution, and then from H to H'.

Moreover, this domain field is related to active/passive connections preoccupations. For self reconfiguration, the modules have to be equipped with connectors that can be locked or unlocked from software. A large field of possible connectors is explored for this particular domain of modular robotics. We can see hooking connections in MTRAN III [7], CONRO [2] and the current Roombot [1] modules, when Molecubes [11], MTAN I and II[7] have magnetic connections. One of the connectors at least has to be active to allow dynamical connection and self reconfiguration. Some systems, like MTRAN modules [7], use female(passive) and male(active) con-

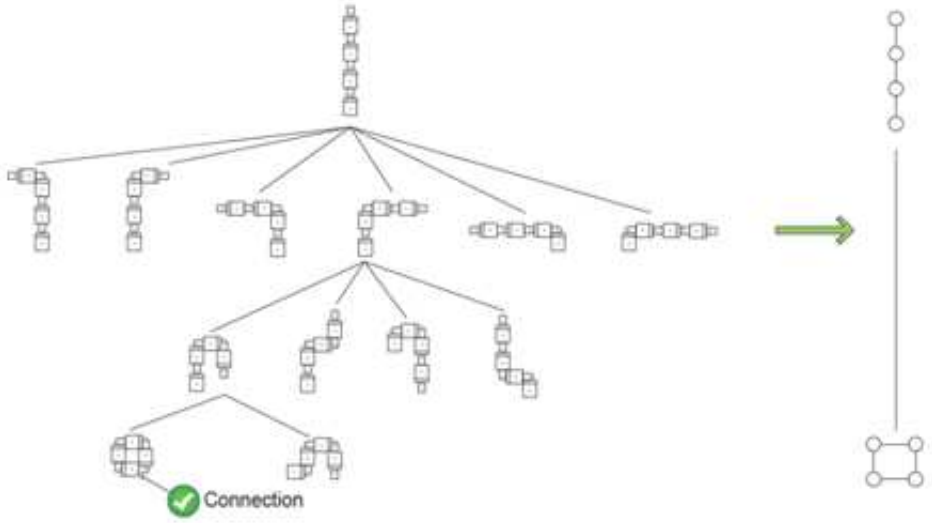


Figure 2.9: An example of the tree of all possible servo movements. Here we use this tree to find which operations leads to a square or circle architecture from a snake architecture. [4]

nectors. Then a female connector can only connect to a male connector, and vis et versa. Other, like the Roombot module [1] use only active connectors, which allow more configuration possibilities.

2.1.3 Locomotion in Modular Robotics

Locomotion in Modular Robotics is also a tough problem. Locomotion in a non-modular robot can be done by inverse kinematics, tuning and optimization over months, but in modular robot we rather want to find some systematic way of learning locomotion gaits, given a robot architecture. The problem is that as the possible robots made by modules are close to infinity (if we put no constraints on size, weight or number of modules), we can't find manually as much locomotion gaits than the possible different architectures. We can then set different ways of resolving this problem:

It is possible to find a given number of locomotion gaits for a reduced number of robot architectures. Then we assume that for being able to move, the robot has to reconfigure itself so that it reaches a configuration that allows locomotion. This has the limitation that the number of modules has to be fixed, or at least that for each number of modules correspond a robot architecture that can allow locomotion.

It is sometimes, particularly when we want to have a biological approach, more interesting to introduce learning in the robot locomotion gait. Learning can be done

online or offline. Offline learning means that a locomotion gait is found by simulation on the computer, and then implemented on the robot with little adaptations. Online learning indicates that the locomotion gait is found directly on the robot: the robot tries in real conditions the locomotion gaits and tries to learn an efficient locomotion gait. Online learning is slower than online learning, because we can't accelerate the simulations on real life. But it has the advantage it allows long life learning, which is very useful if we want to consider the possibility of the robot to adapt when the condition changes. A change can be external (change of gravity, change of the friction coefficient, change of the environment: sand, water, rock, ice) or internal. For example when a module breaks or don't answers anymore, or when the mechanical parameters change over time (fatigue).

Chapter 3

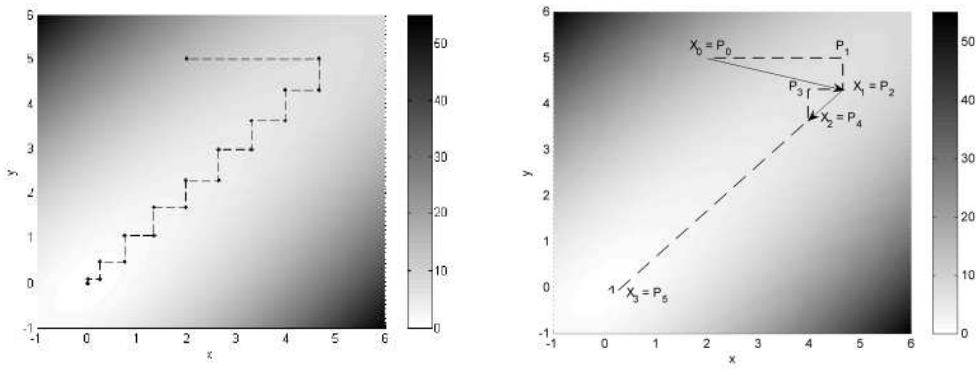
Theoretical Background

3.1 Optimization methods

3.1.1 Powell's Algorithm

Powell algorithm is an optimization algorithm that works in N dimension. Its particularity is that it searches the minimum of a function among a set of directions, and changes this set of direction with new directions for a faster convergence. This is why it is called a set direction algorithm.(Illustrated in Figure 3.1) Powell's algorithm mainly chooses among which direction the minimum is searched. Thus, for finding a minimum among a given direction, we have to use another algorithm. This one is generally a one dimensional algorithm, and can be gradient descendant, but not necessarily. We can mention some examples, as Brent's Method, Golden section search method, and a systematic search method : the function is tested at constant intervals, the final minimum is the minimal value returned.

The principle: Starting from a point P_0 and a base $\{\overrightarrow{b_1}, \ ,\overrightarrow{b_n}\}$ of the domain search (we generally choose for a base the canonical one, where each direction stands for a different parameter to optimize), Powel's algorithm finds the minimum among $\overrightarrow{b_1}$. We call this first point P_1. Then, starting from P_1, it searches the minimum among $\overrightarrow{b_2}$. This is P_2. We iterate n times this steps, and finally find the point P_n, which is the minimum among direction b_n, starting from P_{n-1}. This whole process is one iteration of Powell algorithm. As P_0 is the starting point, and P_n the minimum found after one iteration, we can assume that the vector $\overrightarrow{P_0P_n}$ is more oriented among the gradient vector that the other vectors of the basis $\{\overrightarrow{b_1},,\overrightarrow{b_n}\}$. The idea is then



a) Simple direction algorithm b)Powell's algorithm

Figure 3.1: A famous example where Powell's algorithm (b) provides far better results than a classical algorithm (a). By changing the direction among which the function is minimized, Powell's reach the minimum of the function in only a few iteration, where a classical method needs about ten iterations. [6]

to replace a vector of this basis by this new vector $\overrightarrow{P_0P_n}$, which will ensure faster convergence. Still, 2 questions have to be answered: which vector do we want to replace? And is it always pertinent to replace a vector by $\overrightarrow{P_0P_n}$? We can measure the efficiency of all directions (vectors of the basis), by measuring the decrease of the function among each vector during one iteration. We could intuitively assume that the vectors among which the function decreases the less are the less efficient ones and that we want to replace them by the new vector $\overrightarrow{P_0P_n}$. But this will lead to a base where all vectors are very efficient ones, and thus are quite collinear. On contrary, we will replace the base vector that corresponds to the bigger decrease of the function by the new vector. Like this the base remains highly non-collinear while ensuring fast convergence. The second point is to determine if the new direction is worth being introduced in the base or not. Indeed, by definition, the direction $\overrightarrow{P_0P_n}$ has been the direction among which the decrease of the function has been the largest, as this decrease is the addition of the decreases among all directions of the base. It is thus the more efficient direction for this iteration. But the question is to know if it will remain the best direction for the next iteration. If it is not the case, it will not be useful to include $\overrightarrow{P_0P_n}$ in the base vectors. So we evaluate the function at another point among the same direction $\overrightarrow{P_0P_n}$, but further (Generally, we evaluate the point $\overrightarrow{P_0} + 2 * \overrightarrow{P_0} - \overrightarrow{P_n}$). If the function continues decreasing after the point P_n among the direction $\overrightarrow{P_0P_n}$, then it is worth adding the vector $\overrightarrow{P_0P_n}$ to the basis. Otherwise the next iteration will proceed with the same base vectors as the previous one.

3.1.2 Golden Section Search Method

This algorithm is a one-dimensional algorithm that finds the minimum of a function by bracketing it in a smaller interval at each iteration. If this method is very simple to

Algorithm 1 Powell's Optimization

Require: A starting point P0 and a base of direction $\{b1, ..., bn\}$

loop
  We set Dmax = maximal decease among one direction at 0
  for i=1 to n **do**
    find the minimum among direction bi from point Pi-1, using a 1D algorithm
    if $f(Pi) - f(Pi-1) > Dmax$ **then**
      Dmax=f(Pi)-f(Pi-1)
      d=i
    end if
  end for
  if $PN - P0 < \epsilon$ **then**
    The minimum is reached. $f_{min} = f(P_N)$
  else
    Test the point $P_{extrapolated} = 2 * \overrightarrow{P_N} - \overrightarrow{P_0}$
    t=$2 \cdot (f(P_0) - 2 \cdot f(P_N) + f(P_{extrapolated})) \cdot (f(P_0) - f(P_N) - D_{max})^2 - D_{max} \cdot (f(P_0) - f(P_{extrapolated}))^2$
    if $P_{extrapolated} < P_0$ and $t < 0$ **then**
      find the minimum among new direction $P_{N'} = \overrightarrow{P_0 P_N}$
      $\overrightarrow{b_d} = \overrightarrow{b_n}$; $\overrightarrow{b_n} = \overrightarrow{P_0 P_N}$
      $P_0 = P_{N'}$
    end if
  end if
end loop

implement, it however requires having an initial bracket than enclose the minimum. Golden section search method is thus applied in two steps: bracket a minimum and reduce the interval between the brackets to a given tolerance. It is important to keep in mind that this method is gradient descendant: Is does not ensure that the minimum bracketed is a global minimum.

Bracketing a minimum : The first step of the algorithm is to find three points A, B and C such that $f(A) > f(B)$, $f(C) > f(B)$. This ensures that there is a minimum of $f()$ between point A and point B (which is not necessarily point C). This step is very important, because without these three points we cannot start the Golden search method. We proceed as follow to find these three points: two first points A and B, such that $B > A$, are evaluated. Knowing now $f(A)$ and $f(B)$, we deduce the direction of the slope. If $f(A) > f(B)$, we search a point $C > B$ such that $f(C) > f(B)$. We have then our three points that satisfy the conditions required. If $f(A) < f(B)$, we rename B as C, and A as B. We then search a point $A < B$ such that $f(A) > f(B)$. For finding the third point, we evaluate the function "downhill" at a 'random' point, then increasing the stepsize at each step, until we find a point that is higher than the previous ones. By increasing the stepsizes for each new point evaluated, we increase the probability to find the point we are looking for very fast. This is important because the function is very long at evaluating, and thus the fewer points are needed to try, the best.

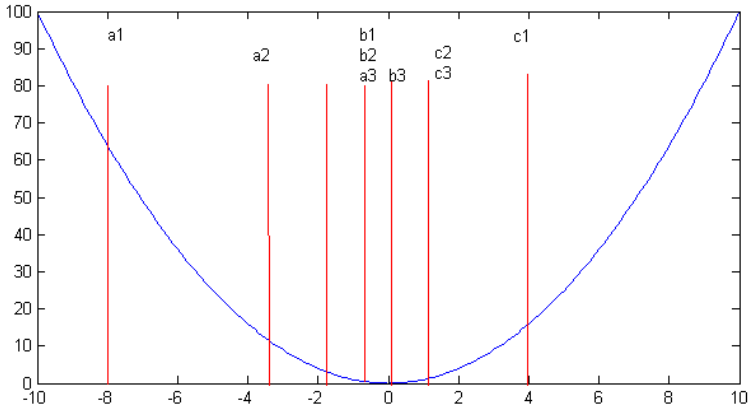


Figure 3.2: Three iteration of Golden Section Search algorithm. At each iteration, we have three points a_i, b_i and c_i, such that $f(b_i)$ is lower than both $f(a_i)$ and $f(c_i)$.

Golden Section Search algorithm : Once we have the minimum bracketed, we proceed to the golden section search algorithm, that reduce at each step the distance between the bracket, while always keeping a minimum between the brackets. The condition of end is that the distance between point A and B is smaller than a given error. This method come from the one used when trying to find a root in a function: We usually find two values x_1 and x_2 with opposite sign, and then we evaluate a point

x_3 which is the middle point between x_1 and x_2, then we keep the two values that are the closest and have still opposite sign, and we iterate until the error is no more significant. In our case, we have no root, but three points A, B and C, where $f(B)$ is smaller than both $f(A)$ and $f(C)$. We also change the ratio for finding the new point: instead of 0.5, we chose the golden section $\frac{3-\sqrt{5}}{2} \approx 0.38197$. The next point is then chosen in the biggest segment with a ratio of 0.38197 starting from the middle point(See Figure 3.2 for illustration). The golden number ratio is analytically proven to be the one that converges the faster in the worst configuration. Moreover, is the initial points are distributed with golden section, the distance between brackets is always $D \cdot 0.61803^n$, where n is the number of iterations, and D the initial distance between brackets. In the other case, the ration between the two segments $\overline{AB}$ and $\overline{BC}$ will converge to the golden number along the number of iterations.

Algorithm 2 Bracketing a minimum

Require: Two random points A and B, golden ratio g=1.618034

return three points A, B, and C, bracketing a minimum
compute $f(A)$ and $f(B)$
if $(f(A) < f(B))$ **then**
  permute A and B; permute $f(A)$ and $f(B)$
end if
$C = B + g * (B - A)$
compute $f(C)$
while $(f(C) < f(B))$ **do**
  compute u , the parabolic extrapolation of a, b, c
  if u is between b and c **then**
    compute $f(u)$
    if $(f(u) < f(C))$ **then**
      $A = B$; $B = u$
    else if $(f(u) > f(B))$ **then**
      $C = u$;
    else
      Compute a new u by default magnification
    end if
  else if (u greater than C) **then**
    if $(f(u) < f(C))$ **then**
      $B = C$, $C = u$, compute a new u by default magnification
    end if
  else
    Compute a new u by default magnification
  end if
  $A = B$, $B = C$, $C = u$
end while

Algorithm 3 Golden Section Search

Require: A, B, C three points such that $f(A) > f(B)$, $f(C) > f(B)$

$x_0 = A; x_3 = C$
if $(\overline{BC} > \overline{AB})$ **then**
  $x_1 = B$
  $x_2 = B + (2 - g) \cdot (C - B)$
else
  $x_1 = B - (2 - g) \cdot (B - A)$
  $x_2 = B$
end if
repeat
  if $(f(x_2) < f(x_1))$ **then**
    $x_0 = x_1$; $x_1 = x_2$; $x_2 = (g - 1) * x_1 + (2 - g)x_3$
  else
    $x_3 = x_2$; $x_2 = x_1$; $x_1 = (g - 1) * x_2 + (2 - g)x_0$
  end if
until Stop condition is reached

3.2 Righetti's CPG

This section provides a presentation of the CPG derived by Ludivic Righetti at BIRG, and that is used for the project. For detailed information please read [9].

This model aims to provide a general CPG structure that can be adapt to all sort of different robots. Il also allows coupling between oscillators and includes sensor feedback. It is particularly adapted to quadruped robots, as it is inspired from quadruped animals, and uses some of their properties like a different duration of stance and swing phases.

The stance and swing phases are respectively the ascending and descending phase of a quadruped leg, or of the oscillator, in the case of the CPG. It was indeed observed that "the speed of locomotion in quadruped animals is controlled by the duration of the stance phase (there exists a linear relation between inverse of stance duration and speed). On the other hand the duration of the swing is almost constant for any gait and has no relation with the speed of locomotion, certainly for stability issues."[9] Living quadruped are a model we want to imitate, so the CPG has to be able to have different stance and swing durations.

This CPG has also strong stability properties, as it has a limit cycle behavior, described by two variables x and y. The output of the CPG is x and can be directly used as the output of the servomotor. As the Roombot module can also work as a wheel (permanent rotation of the motor), we also use for rotation the angle defined

by (x,y) as an output.

Moreover, this CPG is specially adapted for quadruped robots because it allows changing the gait of the robot very easily, only by changing the coupling matrix of the system. The coupling matrix allows to have difference of phases of 0, $\pi/2$ or π. This reduces the search space, as we cannot obtain an arbitrary difference of phase between two oscillators, but actually these three values are sufficient to generate walk, trot, bound and pace gaits. The coupling matrix system will be developed further.

Equations :

$$\dot{x} = \alpha(\mu - r^2)x - \omega y \tag{3.1}$$
$$\dot{y} = \beta(\mu - r^2)y + \omega x \tag{3.2}$$
$$\omega = \frac{\omega_{stance}}{e^{-by}+1} + \frac{\omega_{swing}}{e^{by}+1} \tag{3.3}$$
$$r^2 = x^2 + y^2 \tag{3.4}$$

The stable solution for this dynamical system ($\omega = cst$) is:

$$x = \sqrt{\mu} \cdot \cos(\omega \cdot t) \tag{3.5}$$
$$y = \sqrt{\mu} \cdot \sin(\omega \cdot t) \tag{3.6}$$

Indeed by replacing this expression in the dynamical equation we find:

$$\dot{x} = \alpha(\mu - \mu)x - \omega\sqrt{\mu} \cdot \sin(\omega \cdot t) = \sin(\omega \cdot t) = \frac{dx}{dt} \tag{3.7}$$
$$\dot{y} = \beta(\mu - \mu)y + \omega\sqrt{\mu} \cdot \cos(\omega \cdot t) = \cos(\omega \cdot t) = \frac{dy}{dt} \tag{3.8}$$

This equations describe however a cycle. The and terms are here for stabilization issue: if x and y are not of the form describes early, the dynamical system would make them converge to this form. That's why we call it limit cycle behavior: wherever is the initial starting point of this dynamical system, it always converges to the limit cycle, as illustrated in Figure 3.3. We can choose the radius of the limit cycle by the parameter, and α, β are parameters that control the speed of convergence on x and y axes respectively.

We can also see that ω is not necessary constant. It can depend on two values, ω_{stance} and ω_{swing}. For b sufficiently big, we have

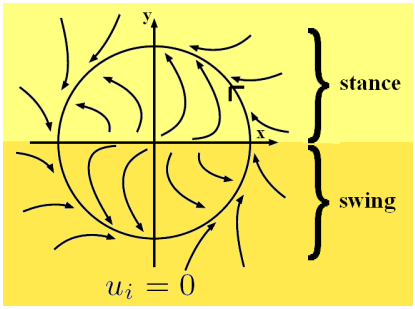


Figure 3.3: The limit cycle behavior of the CPG. If a perturbation sends point (x,y) out of the limit cycle, it would converge to it again, following the arrows.[9]

$$\omega \cong \begin{cases} \omega_{stance} & if \quad y > 0 \\ \omega_{swing} & if \quad y < 0 \end{cases}$$

the speed of rotation is thus determined by ω, and has a different value depending on the sign of y (see fig 3.3). A different speed for the ascending and descending oscillation x has thus been introduced by this equation.

Coupling We add a coupling effect between different oscillators by modifying the second equation [number of equation]. Equation 3.2 is replaced by:

$$\dot{y_i} \quad = \quad \beta(\mu - r_i^2)y_i + \omega x_i + \sum_{i \neq j} k_{ij} \cdot y_j \tag{3.9}$$

We can write the k_{ij} elements in a coupling matrix, also called gait matrix. The diagonal elements of the matrix are 0. The choice of the k_{ij} elements of the matrix determines the coupling between oscillators and the final gait of the robot.

Generally, we determine by a net observation the desired gait. For example, a quadruped walk net is obtained by a difference of phase of $\pi/4$ between legs 1, 4, 2 and 3 (see illustration 3.4), when a trot net is obtained by a difference of phase of 0 between legs 1 and 4, 2 and 3, and a difference of phase of π between leg 2 and 3 (See illustration 3.4)

The theory and its applications for generating our gaits matrix is explained in [9]. As in this project we are applying the CPG, we will keep a practical understanding on this gaits matrix. First we will give the matrix found in [9], as we use the results

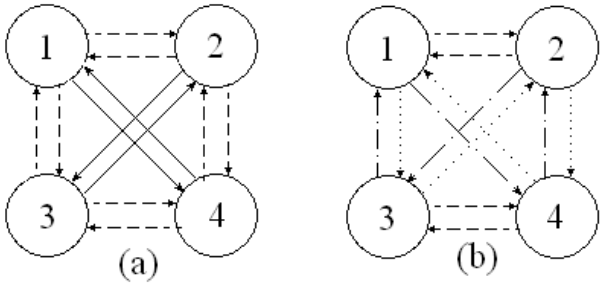


Figure 3.4: Trot gait(a) and Walk gait(b) on a quadruped animal. Each different type of arrow stand for different coupling between the legs.

of this paper. Then we will give some quick tricks to understand how the indices are chosen in this particular application (4 coupled oscillators).

Gait Matrix The 4 gaits of [9] will be implemented on our table robot. They allow generating 4 different common gaits of quadruped animals, which are the trot, pace, bound and walk(Figure 3.5).

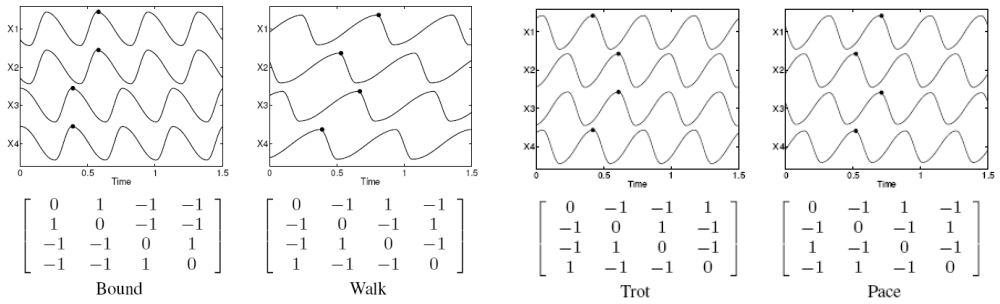


Figure 3.5: Coupling matrices and examples of gait generation for the 4 gaits. $\omega_{stance} = 2 \cdot \omega_{swing}$ for the trot and pace gaits, $\omega_{stance} = 4 \cdot \omega_{swing}$ for the walking gait and $\omega_{swing} = 2 \cdot \omega_{stance}$ for the bound.[9]

When putting in parallel the network of trot and the corresponding matrix, we can first analyze two oscillators i, j with respect to their respective indices in the gait matrix and the difference of phase between them:

k_{ij}	k_{ji}	δ phase
1	1	0
1	-1	$\frac{\pi}{2}$
-1	1	$\frac{-\pi}{2}$
-1	-1	π

More generally, we can deduce than a $k_{ij} = 1$ indicates the same phase between the oscillators, and an indice $k_{ij} = -1$ indicates opposite phases. When for two

oscillators i and j, i tries to have the same phase than j, and j tries to have the opposite phase than i, at equilibrium the result is that they tend to have a difference of phase of $\pi/2$.

Sensor Feedback This CPG allows also introducing sensor feedback. For this, we replace Equation 3.2 by this new one:

$$\dot{y_i} \quad = \quad \beta(\mu - r_i^2)y_i + \omega x_i + \sum_{i \neq j} k_{ij} \cdot y_j + u_i \tag{3.10}$$

where u_i is defined as

$$u_i = \begin{cases} -sign(y_i) \cdot F & fast\ transition \\ -\omega_i \cdot x_i - \sum i \neq j k_{ij} \cdot y_j & stop\ transition \\ 0 & otherwise \end{cases}$$

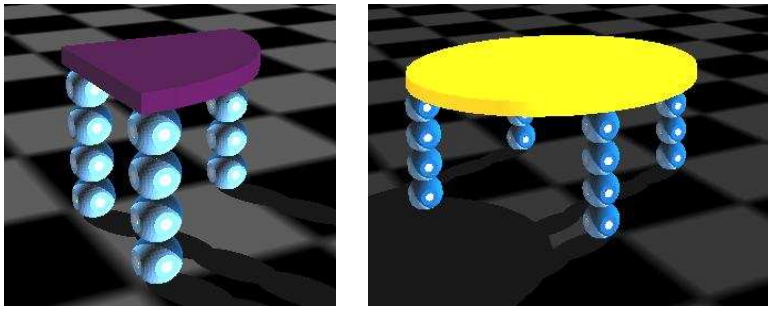
For this project sensor feedback were not implemented, as the actual Roombot module have no sensors. It would however be interesting to see the effects of sensors Feedback on the results for locomotion. This point will be discuss in chapter 6.

Chapter 4

Methodology

4.1 Design of the robots

This project takes place in a bigger field of research in modular robotics. It is linked to others project that have taken place at BIRG. The Roombots were initially design to be used as intelligent pieces of furniture that can reassemble in other pieces of furniture, and can move. For other aspects of the Roombot projet, one can read [4]. Initially, the passive elements were chosen to be modular as well, and the roombots pieces of furniture had real size: a height of 45 cm for a chair or stool, and 75 cm for a table. Finally, as the Roombot modules are not able to support one person's weight, and for stability of the robot's locomotion gaits, experiments were done on modified robots, and concentrate only on the locomotion part, and not on the size and design of the robots. The two final models that were tried are those of Figure 4.1.



a) chair robot b) table robot

Figure 4.1: Two robots that were used for the simulations. Initially, robot a) was designed as a chair and robot b) as a table. Finally they don't have good dimensions, neither sufficient torque to be used as real pieces of furniture.

4.2 The Roombot module and its free parameters

The Roombot module is largely inspired from the Molecube module [11]. It is composed of two cubes-like. The main difference with the Molecubes is that a degree of freedom is added between the two cubes. Its external dimensions are 220x110x110mm, it weights 1.2 kg and it has three degrees of freedom, as we can see in Figure 4.2. The axes A and B are along the longest diagonal of the cubes. The three degrees of freedom are controlled by 3 servomotors. Each one will have in the simulation a maximal torque of 4 Nm. As we will see in the discussion section 6, the choice of the torque is critical: it has to be as high as possible, given the dimensions of the module. The modules have on each free face of the cubes a connector for connect with other modules. There are 10 connection per module.

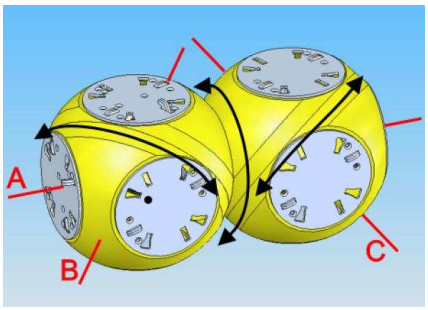


Figure 4.2: The Roombot module, with its three axes of rotation A, B and C

4.2.1 Free parameters and their influence

For illustration, let say we search the number of free parameters that we can set, for a given four-egged robot, each leg made of 2 modules. (Example of the table, see fig 4.3).

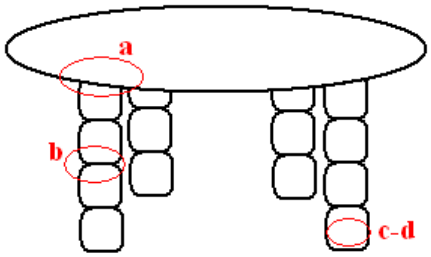


Figure 4.3: This chapter will take as an example this four-legged robot. We make the distinction between the different parameters: a) Connection between modules and passive elements, b) connection between different modules, c)servo parameters and d) CPG-model parameters.

For this example, and given the first structure constraint, we can choose parameters at different levels:

1. The connection between the passive element and the modules
2. The connection between different modules
3. The servo command parameters
4. The CPG parameters

The way the modules are connected at the table defines the direction of the 1st axis of the module, and influences directly the global behavior of the robot. There are 4 main ways to connect the different modules to the table, as it is illustrated in Figure 4.4, which correspond to 4 different motion models.

Figure 4.5 shows the 3 different ways to connect a module to another, without losing degrees of freedom: The two axis can be parallel, perpendicular or skew. The kind of connection constraints the motions of the leg. For a better understanding, Figure 4.5i) shows the two connected parts as a single element, and how the rest of the modules can move, depending of the type of connection. We can therefore see that they lead to three different possible movements of the articulation.

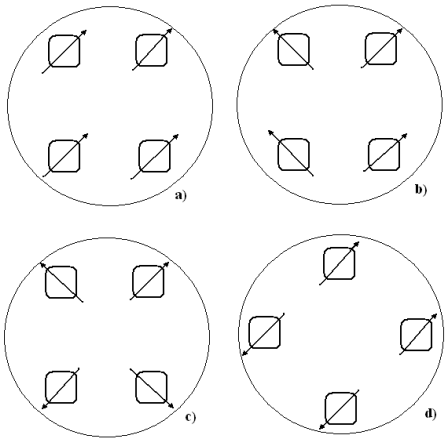
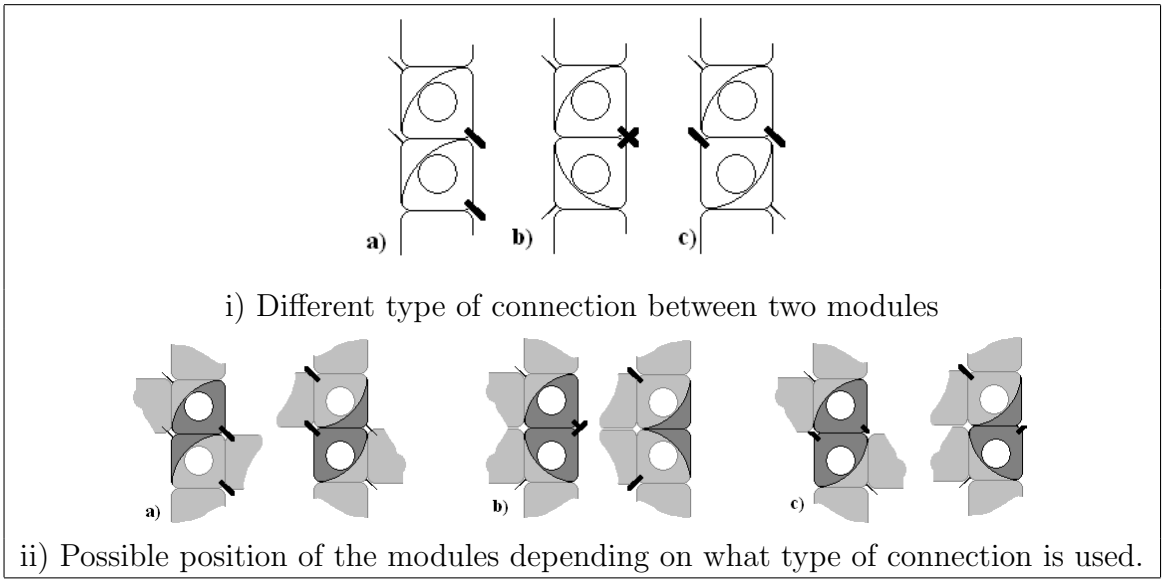


Figure 4.4: Four examples of connecting the modules to the passive element. Arrows represent the axis of rotation of the first servo motor.

4.2.2 Command of the servo

Each servomotor is given an angle as a command. The command of the servo cans be in permanent rotation (like a wheel), or in oscillation. The command is different



i) Different type of connection between two modules

ii) Possible position of the modules depending on what type of connection is used.

Figure 4.5: The three type of connection between two modules are parallel (a), perpendicular (b) or skew (c).

depending on which mode the servo is working. Here are the command line of a servo, with its different parameters:

$$\alpha = \begin{cases} \pm(\arctan\left(\frac{x}{y}\right) + offset) & if \quad rotation \\ \pm(x + offset) & if \quad oscillation \end{cases}$$

x and y are given by the CPG model. So the free parameters remaining are the offset, the sign, and what we call the R/O parameter, which sets if the servo is rotating or oscillating. The sign determines the sense of rotation for a servo that works in rotation. It determines for oscillations in which direction the servo is swinging, and in which it is standing. This parameter is mostly to assure the symmetry of the motions. It depends on the referential of each servo (Where is the degree 0, in which direction are the positive angles), and also on the direction desired for a given displacement, or for a given motion of the leg. The offset: for oscillations it determines around which point the servo oscillates, and for rotation, it sets from which angle the servo is in swing, and at which angle it is in stance. As a recall, stand and swing phases differ in their angular speed.

As seen at page 16, the equations of the CPG are as following:

$$\dot{x} = \alpha(\mu - r^2)x - \omega y \tag{4.1}$$
$$\dot{y} = \beta(\mu - r^2)y + \omega x \tag{4.2}$$
$$\omega = \frac{\omega_{stance}}{e^{-by}+1} + \frac{\omega_{swing}}{e^{by}+1} \tag{4.3}$$
$$r^2 = x^2 + y^2 \tag{4.4}$$

α,β and b are intern parameters of the CPG. They coordinate the speed of convergence on the limit cycle of x,y. In this work we will take them same as suggested in [9], i.e. $\alpha = 5$, $\beta = 50$, and $b = 100$. $\sqrt{\mu}$ is the radius of the limit cycle. It can also be seen as the amplitude of the oscillations if the servo works actually as oscillator. It has no influence if the servo works as a wheel. ω_{stance} and ω_{swing} are related to the angular speed for two phases of the oscillation or rotation. This allow the oscillation to have different slopes for ascendant(stance) or descendant(swing) phase of x. For illustration of this different parameters see Figure 4.6

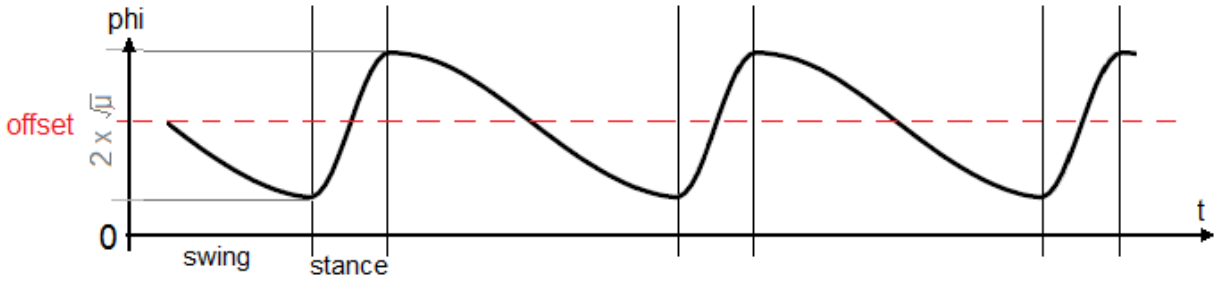


Figure 4.6: An illustration of the servomotor parameters w_{stance} and w_{swing}, offset and amplitude of the signal

Last parameters to set are the k_{ij}. This ones are indices of the coupling matrix, also called gait matrix. The gait matrix is strongly related to the difference of phase between two oscillators. k_{ij} can be equal to 1 for same phase between oscillator i and oscillator j, -1 for opposite phases and 0 for no coupling between oscillator i and j.

4.3 Application of the CPG on the robots

We have seen in the previous chapter that there were a lot of free parameters for a given robot structure. Some of these parameters are continuous, like an amplitude of an oscillation(from 0 to π), and other are discrete, like the type of connection between two modules (parallel, orthogonal or skew). We use Powell's optimization algorithm for finding a motion model that optimizes the radial distance covered in a simulation run. So as Powell's algorithm is gradient descendent, it can only optimize continuous parameters. Moreover, if we want to found an optimum in a relatively short time (a few hours), we have to reduce the parameters to optimize to less than 10. For more parameters, the computation time will be too long for envisaging online learning. If we take the same example than before (the table structure, 4 legs and 2 modules per leg) we can calculate the number of free parameters:

Nature of the parameters	Analytic expression	Number of param
Connections between modules	$L \cdot (M-1)$	4
CPG parameters (ampl, w_{stance} and w_{swing})	$3 \cdot 3 \cdot M \cdot L$	72
Servo command parameters (sign, offset, Rotation/Oscillation))	$3 \cdot 3 \cdot M \cdot L$	72
Connexions between the passive element and the modules		1
Gait matrix (phase between the different servomotors)		1
Total		**150**

(M=number of modules per leg, L= number of legs)

Moreover, it is very important to remember that the discrete parameters have to be set arbitrary by the user, or tried: for each possible permutation of the discrete parameters, the optimization has to be run, and finally we choose the minimum of all minimums find for each optimization with a given set of discrete parameters. The possible combinations of discrete parameters are quite numerous: $\sum_i num\ parameter(i) \cdot possible\ values(parameter\ i) = 74\ Possible\ combinations$

We are going to test three different methods on three different robots. For the first version of the tripod(Figure 4.1a), we have made some hypothesis to reduce the number of parameters:

- Each module oscillates / rotates at the same frequency: w_{stance}, w_{swing} are the same for all servomotors.
- Symmetry of the legs: all legs have the same distribution of offset and amplitude among the legs. (divides by L the number of offset and amplitude parameters)
- The distance covered by the robot is proportional of the frequency of oscilliation/rotation. We can set the frequency at an arbitrary value (for example $w_{stance} = cst$), and optimize the ratio $\frac{w_{stance}}{w_{swing}}$ instead of optimizing the two parameters w_{stance} and w_{swing}.
- The two last servomotors of each leg can always be set at a 0 value. They don't contribute to the motion gait.

This leads to the remaining free parameters:

Continuous parameters	number
offset	$4 = (M \cdot 3 - 2)$
amplitude	$4 = (M * 3 - 2)$
w_{swing}/w_{stance}	1

Discrete parameters	number
sign	$16 = (M * 3 - 2) \cdot L$
Rotation/oscilliation	4=(M*3-2)
Passive/modules connection	1
Module/module connections	$4 = (M - 1) \cdot L$
Gait Matrix	1

We can see that only 9 free continuous parameters need now optimization. For the tripod optimization, we reduced it to 7 parameters by setting an additional symmetry between modules: All modules have the same amplitude and offset repartition among the servomotors. We can also try other methods, like chose three servomotors among the legs that will rotate or oscillate, the other servo being considered as solid connections. Another way of reducing the number of parameters is up to the imagination of the user. We can note that reduce the number of parameters from 9 to 7 can seem not very useful, but it reduces considerably the time for optimization, and these techniques have the advantage of reducing the number of parameters to 7 whatever the number of modules the robot has.

The second tested robot was the four-legged robot, with a different strategy. Here the goal is to test the different gaits of [9] on the table robot. The first step was to find a leg motion gait that would have same behavior as mammal legs (Figure 5.6, page 32). The leg motion gait has no open parameter, as the pattern of the gait is defined. The three open parameters will be the angular speeds of swing and stance, and the maximal torque value. Results are obtained by systematical search on the angular speeds, for different values of maximal torque.

The third robot tried is an often used gait with tripods which looks like a swimming guy, with two arms at the front that push the robot (also like little turtles in the sand), and a tail that helps pushing in the back (see Figure 4.7). We used the tripod robot, but as a consequence of the results for the first experiments (see section 5.1), the size of the passive elements is twice as large as the previous one. We assume this would be a good solution against the lack of stability of the first tripod robot.

In this case we have symmetry of the two arms while the single leg or tail has its own motion model. In this case we would have 17 open parameters. As we mentioned it, it takes too much time to optimize more than 10 parameters. The strategy for optimizing this gait would then to make the simulation in two steps: First the two fore limbs are optimized, which leads to only 8 parameters to optimize, and provides a locomotion gait where the tail is not moving, and the robot is crawling with two limbs. Once the two forelimbs' gait have been optimized, the tail optimization can be started, leading to 9 more open parameters. This strategy assumes that the parameters of two servomotors that are not in the same limb are not directly related. Which is quite a good assumption, as we mostly want to find a limb's gait that allows the robot to move.

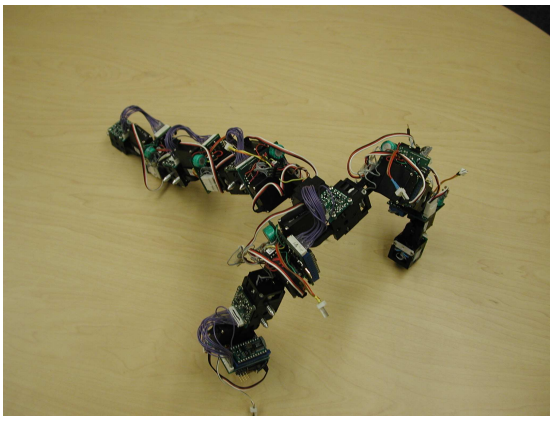


Figure 4.7: CONRO Tripod Robot: Two legs are symmetric and are used for locomotion. The third one is like the prolongment of the body and is useful because it increases the stability of the gait. [2]

The results for this experiments are not written down on this report, but will be presented and discuss in the final defense of the project.

4.4 Computation of the fitness function

Golden section search algorithm and Powell's optimization are implemented for finding a minimum of a function. The goal of this project is to find some efficient gaits for straight locomotion. We would thus maximize the radial distance covered in a simulation run.

The idea is then to found a mathematical expression $f(D)$, where D is the radial covered distance, such that $fit(D_{max}) = fit_{min}(D)$. The fitness function used for the results of the tripod is: $fit(D) = \frac{1}{1+D}$.As the distance is always positive, this fitness function is limited between 0 and 1. This is the main advantage of this fitness function: Indeed, as the parameters are restricted within a range of possible values (between 0 and 2π for an offset on the motor command for example), it is easy to replace the fitness value at a point P out of the range by a number bigger that 1. For example, we can set $fit(D) = 2$ if P out of range, or $fit(D) = 1 + ||P||$ if P out of range. However, the second expression is better, because it creates a slope in direction of the search space, like shown in Figure 4.8

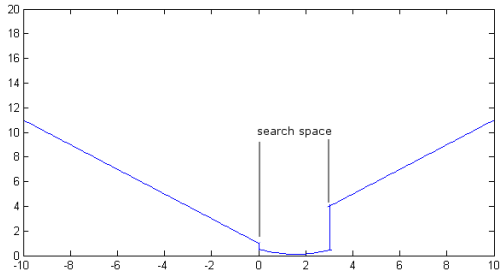


Figure 4.8: The fitness function for a limited search space is extended to an infinite search space by replacing $f(P) = \frac{1}{1+D}$ by $f(P) = 1 + ||P||$ if P is out of range

Chapter 5

Results

5.1 Three-legged Robot

For the tripod robot, we searched a motion gait with 7 open parameters. As we have seen in the chapter 4.2, there are a few different ways for reducing the number of open parameters for an optimization search.

Here are the results for the first described distribution of open parameters. We made Powell Optimization on 7 open parameters, which are the offset and amplitude of each servomotors of 1 module, and the wstance/wswing parameter. So each module has the same motion model. The max torque of each motor was set to 4Nm. The simulation lasts 20 seconds and the minimized value is $\frac{1}{(1+D)}$, where D is the radial distance covered during the trial. The different legs where not coupled together.

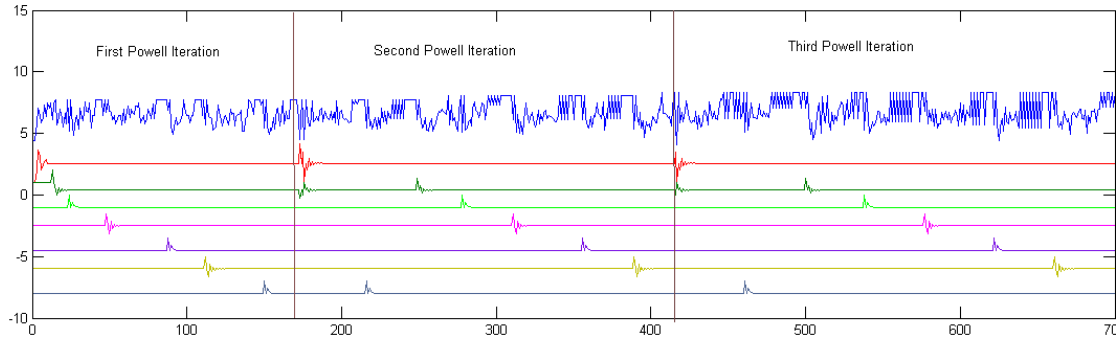


Figure 5.1: Iterations of Powell's optimization for modules that works in oscillation. The first line is the distance covered in a simulation run, the 2nd to 8th lines are the 7 open parameters. After optimization, $D_{max} = 4.30$

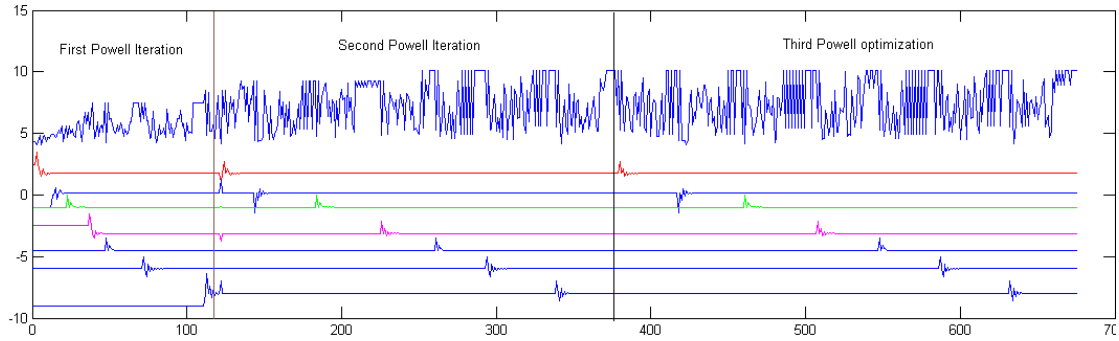


Figure 5.2: Iterations of Powell's optimization. Motor 3 works in rotation while the others are in oscillation mode. The first line is the distance covered in a simulation run, the 2nd to 8th lines are the 7 open parameters. After optimization, $D_{max} = 6.11$

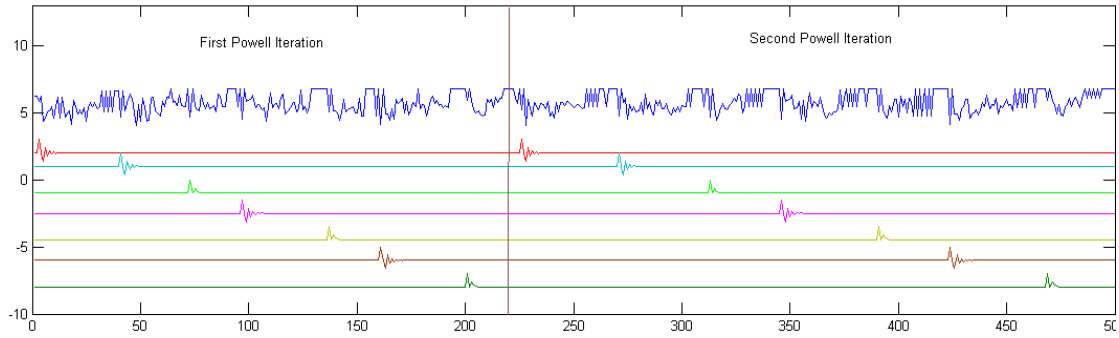


Figure 5.3: Iterations of Powell's optimization. Motor 2 works in rotation while the others are in oscillation mode. The first line is the distance covered in a simulation run, the 2nd to 8th lines are the 7 open parameters. After optimization, $D_{max} = 2.80$

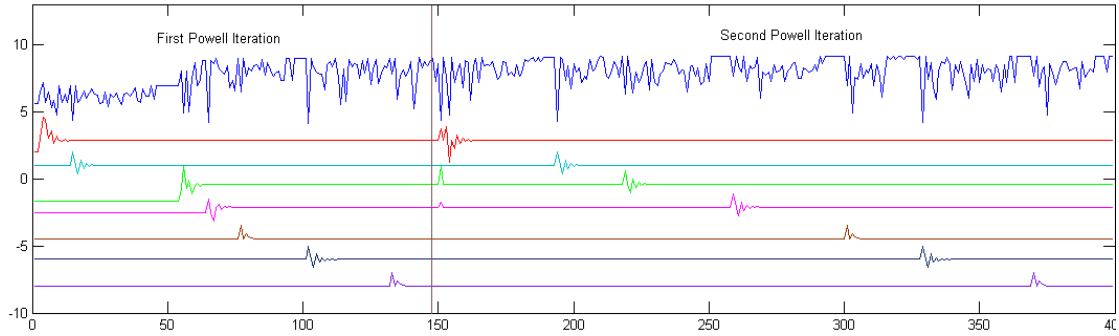
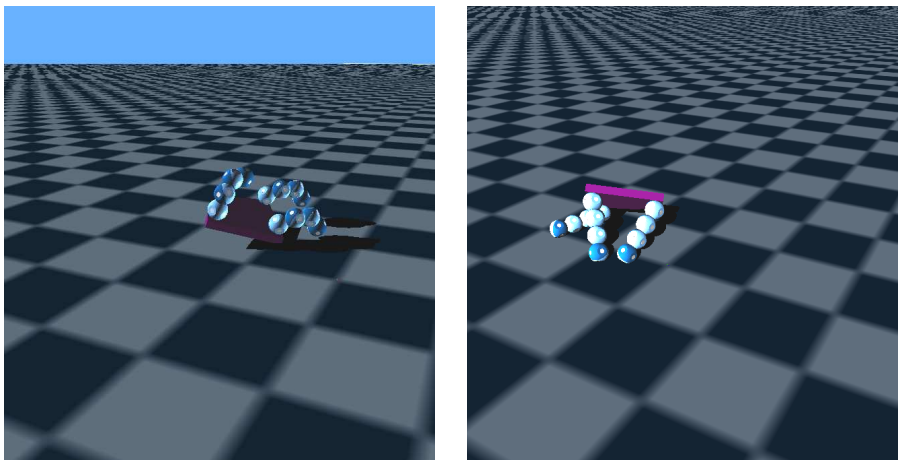


Figure 5.4: Iterations of Powell's optimization. Motors 2 and 3 works in rotation while the others are in oscillation mode. The first line is the distance covered in a simulation run, the 2nd to 8th lines are the 7 open parameters. After optimization, $D_{max} = 5.14$

We can thus see in graph above that the landscape of the function we want to minimize is very irregular. The problem here is that the value returned is very noisy. This explains why the optimization process gets stuck in a local and very close from starting point minima. It is clear that here we have a problem of stability, as in reality so little changes of a parameter would not lead to so big changes of the efficiency of the robot. This noisy behavior is caused by the values of maximum torque used in the simulation. This max Torque value is too low, which means that the robot can't support its own weight, and sometimes collapse and is not able anymore to get up. As this behavior is very unstable (we don't know when the robot will fall), and very sensitive to very small changes of parameters, we obtain such noisy fitness functions. Actually, we tried also some simulations with max torque at 8Nm, but this time the elasticity of the robot made him fall legs up, which is another unstable behavior. Figure 5.5 shows snapshots of the two previously explained instabilities.



a) Robot falling b) Robot collapsing

Figure 5.5: This robot's experiments show a lack of stability of the robot. If the maximal torque allowed in simulation is too high (8Nm), the robot falls. If the maximal torque allowed is too low (4Nm), the the robot can't support its own weight and collapses.

As we see the results of the chair robots, we can make some deductions and decisions for the works on the table robots. First, we use a CPG developed for a quadruped robot. It can of course be adapted to other robots, but as we have seen, the results are not very convincing. First, the gaits with the tripod are unstable: Because of unadapted maximal torque values, and the size of the robot (legs very close) the behavior of the robot is unpredictable, and we can't proceed to any optimization. Then we can see that the application of the CPG is not optimal : only 7 open parameters, all modules have the same command. Moreover no coupling between legs was implemented, as the CPG is initially derived for quadruped robots.

5.2 Four-legged Robot

For the table robot, we decided to take another direction of research. We first search for a motion model of one leg, that acts like a quadruped animal leg. The aim is to have a cyclic movement that is like half a circle (Figure 5.6)

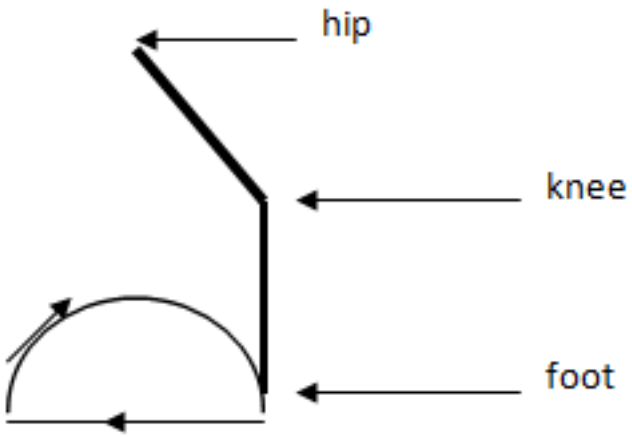


Figure 5.6: A quadruped animal's leg as generally this kind of motion cycle. The same pattern is then searched for our robot.

For finding such a pattern, we could do inverse kinematics or find a solution by hand tuning. The advantage of inverse kinematics is that it provided equations to solve, and is these equations are solvable, then we can find all solutions. Moreover, this equations could lead to more open parameters, allowing optimization.

However, due to the complexity of Roombot kinematics, the solution chosen was tuned by hand . We first tried to have the half circle behavior, and then the flat behavior. We then combined the two behavior, each on half a cycle, to obtain the desired motion model.

The final motion has only 1 open parameter, which is the output x of the oscillators (CPG). It can for example be compared to the motion of a chicken leg. The motions of the first servo is the motion of the hip and, as said before, is controlled by the CPG oscillator, and coupled with the other legs of the robot. The other servomotors are controlled by different functions of x, as illustrated in Figure 5.7.

The corresponding command sent to the motors:

$$\alpha_{s1} = x \tag{5.1}$$

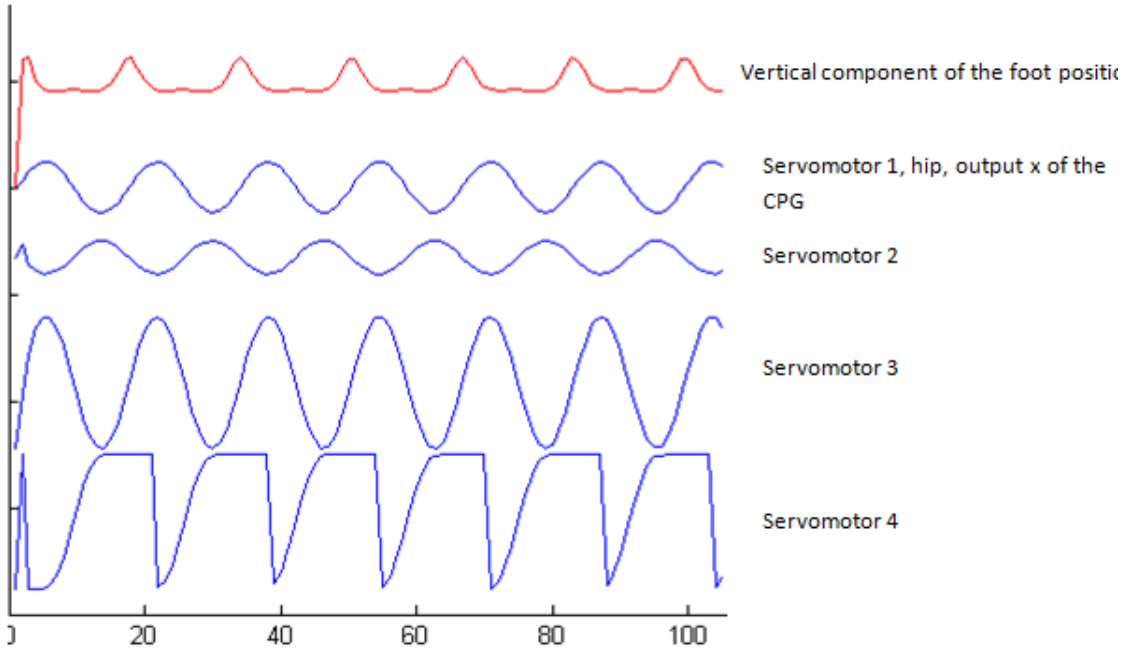


Figure 5.7: The command send to the 4 motors in blue, and the resulting vertical motion of the foot (in red) among time. The desired pattern was reached: The foot remains at the same height during half a cycle and then the leg length bends during the other half cycle.

$$\alpha_{s2} = -\frac{\pi}{8}\frac{x}{\sqrt{\mu}} \tag{5.2}$$

$$\alpha_{s3} = \pi(1+\frac{x}{\sqrt{\mu}}) \tag{5.3}$$

$$\alpha_{s4} = \begin{cases} 2\pi & if \quad y > 0 \\ \pi(1-\frac{x}{\sqrt{\mu}}) & if \quad y < 0 \end{cases} \tag{5.4}$$

Where x, y the output of the CPG, and $\sqrt{\mu}$the amplitude of oscillations of the CPG oscillator.

We implemented this motion equations in the control of the modules, and also implemented the gait matrix provided in Ludo's paper. Then we started a systematical search with w_stance and w_{swing} as open parameter for the four gaits, and with maximal torque at 10 Nm. This Torque are impossible to provide in a real module, but they will be our reference for further measurements with lower torque (We can for example compare the distance covered with the max torque at 4 and 10 Nm , and if the difference is too big, then it means that the motors cannot provide the needed torque and the gait is not efficient.).

The results for the systematical search are shown at Figure 5.8, for a max torque at 4Nm, and a run of 40 seconds. Figure 5.9 shows the maximal speed reached by the robot in different gaits, depending on the maximal torque allowed in simulation.

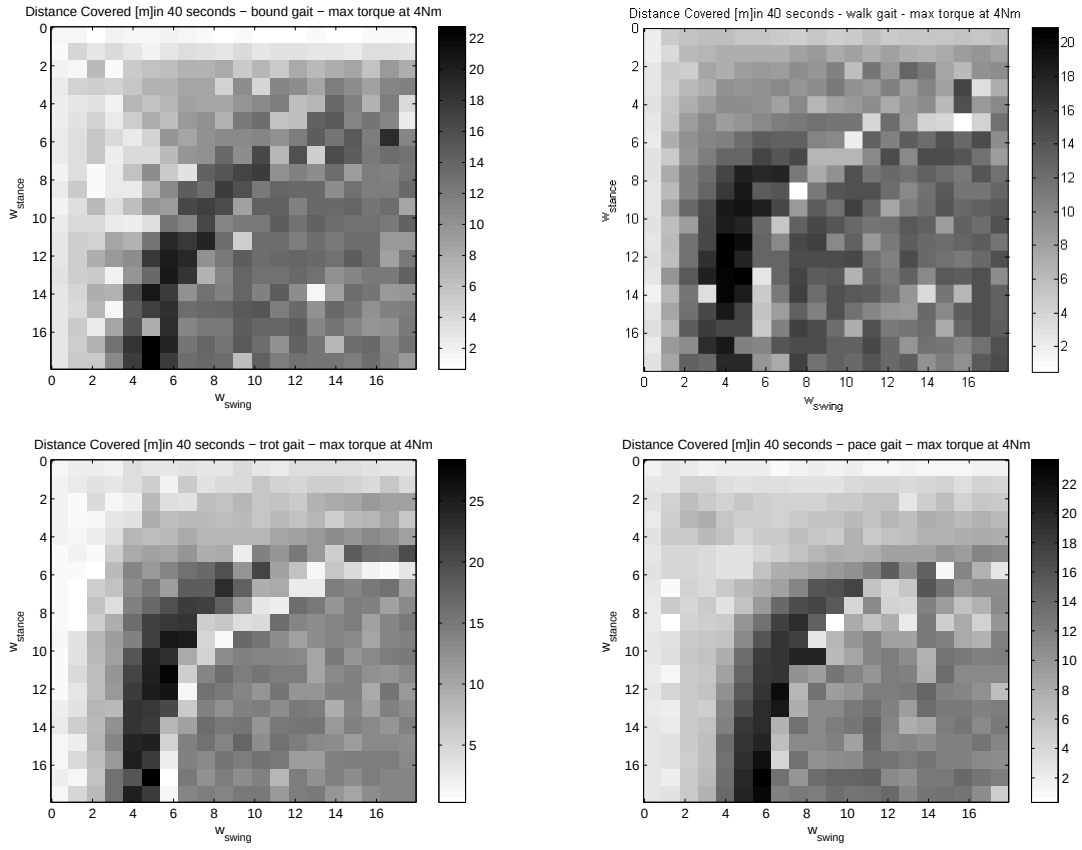


Figure 5.8: Results of systematical search for the different gaits.

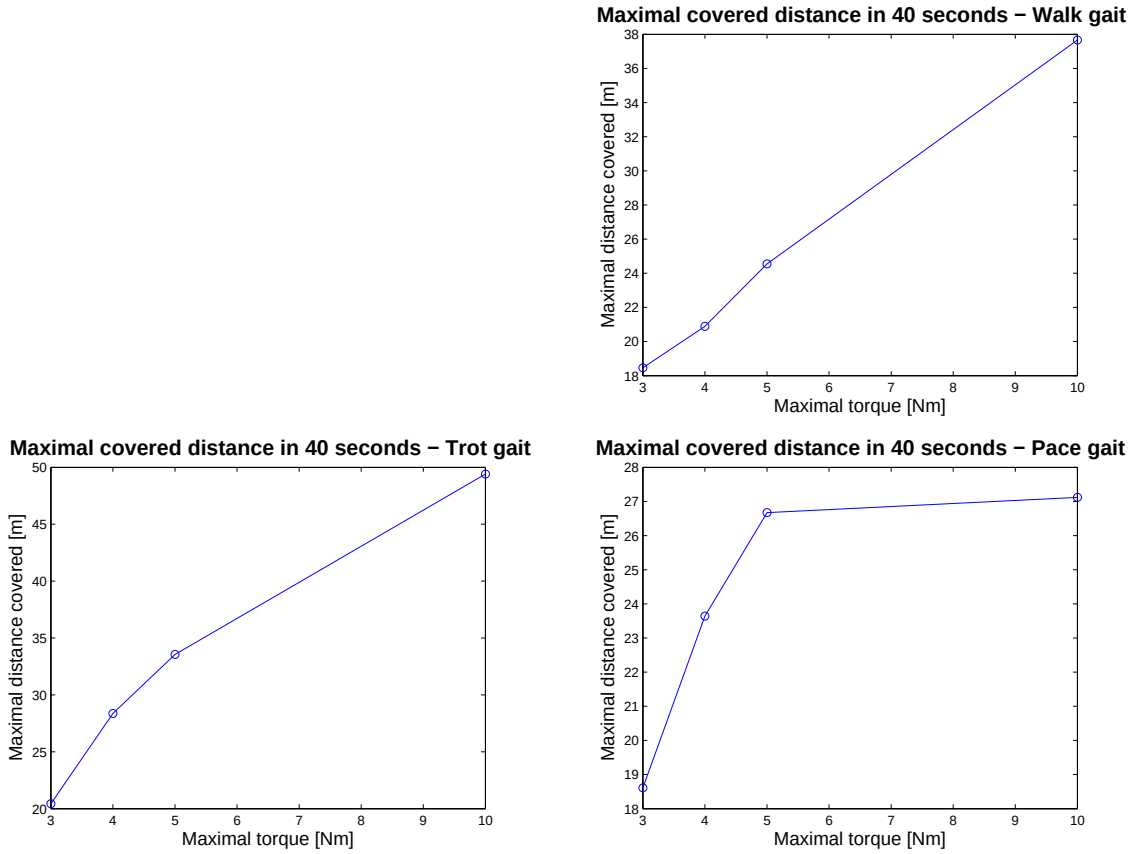


Figure 5.9: Results of systematical search for the different gaits, according to the maximum torque allowed.

Chapter 6

Discussion

6.1 Limitations of Roombots and Webots Simulations

6.1.1 Max torque

As in the physical implementation of Roombots modules, the motor's torque will be limited, it is very interesting to see the influence of lower torque on the efficiency of the gaits. When the maximal value for torque decreases, some motion or positions of the robots that needed higher torque are not possible anymore. This leads to imprecisions or delays in the actions of the motors, and a global gait less efficient. We can see it in Figure 5.9: The maximal distance covered in a simulation run decreases when the maximal torque decreases. But which is also interesting, is that the efficiency of each gait decreases in a different manner: the walk and bound graphs show a linear decrease, where pace and trot gaits seems to have exponential decreases.

Figure 6.1 resumes the behavior of the 4 gaits in the same graph. Trot seems to be the best gait for all max torque values. Pace is more efficient than bound and walk for low torque (4 and 5 Nm), but is the worst gait at 10 Nm. Bound gait decreases linearly, which is quite surprising. We could think that beyond a given torque, the robot is not able anymore to bound, which leads to very bad performances. But these results tend to show that the distance covered in a bound gait is proportional to the max torque allowed in the motors.

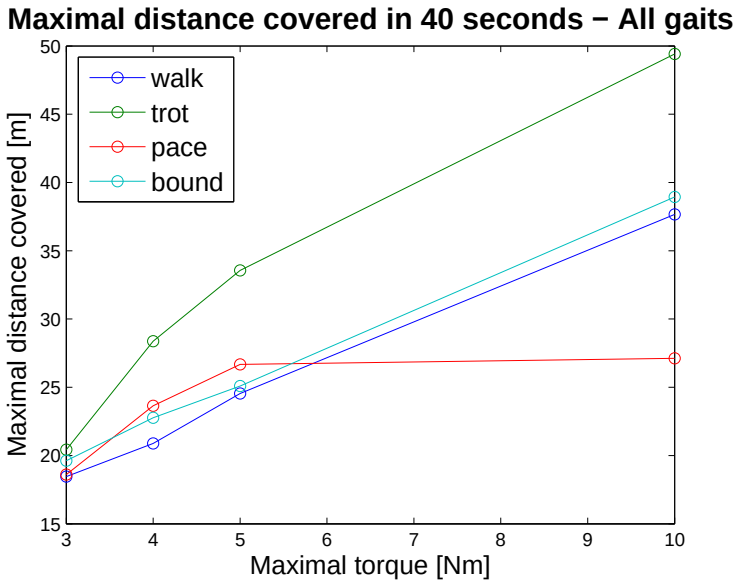


Figure 6.1: The best distance covered for each gait and different maximal torque values. Trot gaits has systematically the best results.

6.1.2 Best performances' areas

The graphics 6.2 show the distance covered for different w_{stance} and w_{swing}. Under some specific max torque value, some areas seems to be optimal, because they concentrate most of the higher values of the graph. We call this area the "best performances' area". Best performances' areas are not located at the same place depending of the gait.

Actually, according to [9], it would be more likely that the speed of locomotion would depend only of the swing duration and not of the stance duration. This would give graphics with vertical lines. But these results show a distance covered that is essentially dependant on the average speed of the servomotors. As we can see in graph 6.3, with a maximal torque at 10 Nm, the distance covered is dependant both on ω_{stance} and ω_{swing}, and is higher for high average speeds of rotation of the motors. Such unexpected behavior could disappear if we add sensor feedback to the robot structure and CPG control. Some experiments with implementation of such sensors will be interesting, as they could lead to a comparison with the results presented here.

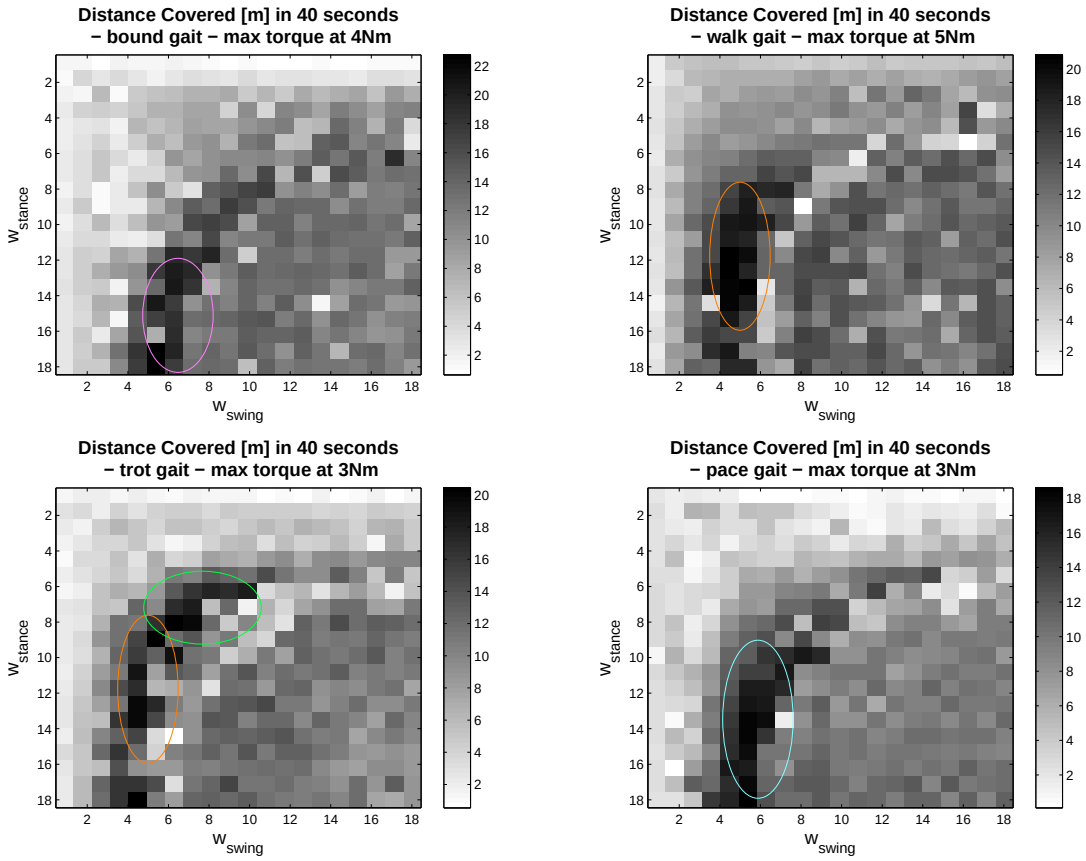


Figure 6.2: The ellipses show the areas of the search space where the distance is maximal. However, according to [9], the plots should have horizontal strips, because the speed of locomotion is not dependant of $w_s wing$.

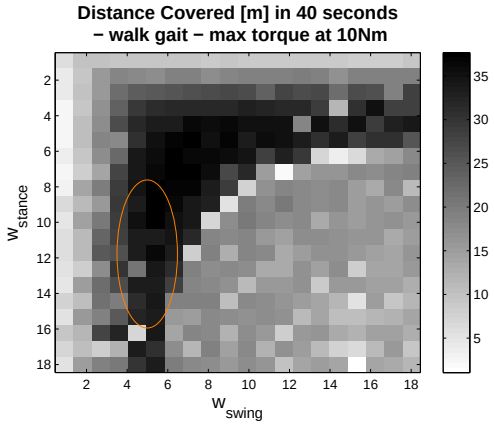


Figure 6.3: Here again, the distance covered in a simulation run seems to be dependant of both ω_{swing} and ω_{stance}.

6.1.3 Max speed

A round pattern shows an area where the distances covered fall abruptly. This pattern corresponds to the maximal speed allowed for the motors. The speed motors

is limited at 10 in the simulation. More generally, if we have a speed motor limited by ω_{max}, and a periodic motion that has two phases T_1 and T_2 at two different speeds ω_1 and ω_2, then the global speed ω is :

$$\omega = \frac{4 \cdot \pi}{T_1 + T_2} = \frac{2}{\omega_1^{-1} + \omega_2^{-1}} \tag{6.1}$$

$$\omega_{1,max}(\omega_2) = \frac{\omega_{max}}{2 \cdot (1 - \frac{\omega_{max}}{2 \cdot \omega_2})} \tag{6.2}$$

In Figure 6.4, the blue curve is the theoretical limit speed, and the red one is the effective limit speed observed during the simulations. We can observe a slightly different ω_{max} in the results that the one expected. This difference can be explained by different factors: If the maximal torque allowed is too low (3Nm), then the effective maximal angular speed decreases: The curve corresponding has a w_max of 3.7 for the trot gait at 3 Nm. But at higher torque this parameter has no more influence: an effective w_max of the trot gait at 4.2 for 4, 5 and 10 Nm. It is also possible that at high speed, close of saturation speeds, the motors are not able anymore to correct errors between the command and the effective position of the motor, as they can't reach the needed speed to correct them. Now coming back to Figure 6.2, the best areas performances can be explained by this speed of saturation. The curves on Figure 6.4 shows the maximal ω , but with no restriction on ω_{swing} and ω_{stance}. In reality, ω_{swing} and ω_{stance} are also limited to this maximal speeds. But, as the motors are commanded in open loop, if there is speed saturation in the stance phase, the speed of swing phase will increase to compensate the delay in the command of the motor. Indeed the frequency of the signal send to the motors is constant, and an error due to saturation in stance phase will be corrected by the regulator P in the swing phase by increasing the speed of motors. This behavior near saturation is illustrated in Figure 6.5. This shows a bigger influence of ω_{swing} than ω_{stance} on the global speed of locomotion of the robot.

6.2 Which algorithm for gaits learning?

This has been the most important question during the project. Indeed, the first experiments on the tripod robot showed a total failure with Powell's algorithm and Golden Section Search Method: The fitness functions are very uneven, as it can be observed on Figures 5.1 - 5.4. So all the time, optimization process is stuck in local minima close from the starting point. When looking at Figure 5.8, the second robot seems to have also an uneven fitness landscape. In such configuration of the fitness landscape, the used algorithm are not efficient anymore. So the idea is to use an

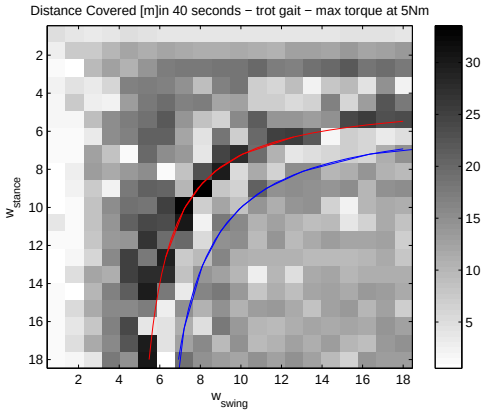


Figure 6.4: The curve of theoretical speed of saturation (in blue) and the effective curve of speed saturation (in red) don't coincide. For this trot gait at 5Nm, the theoretical maximal speed is of 5 rad/sec, but it is observed at 4.2 rad/sec.

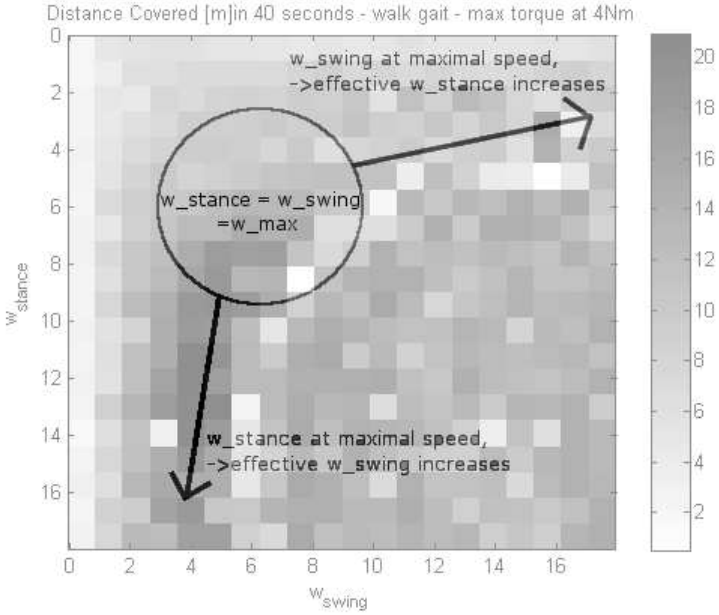


Figure 6.5: When one of the speeds reaches its maximum, the other speed increases for compensate, as illustrated.

algorithm less dependant of local minima. Powell's remains a good solution if we replace the one dimensional optimization method. Using for example a systematical search instead of Golden Section Search Algorithm.

Other solutions would be to use GA (Genetic Algorithms) or Particle Swarm Optimization. The problem of this two algorithms is that they need too much evaluations of the fitness function. For each evaluation of the fitness function a 20-seconds simulations has to be run, which is costing in time. Generally Particle Swarm Optimization needs around 20 particles, which leads to 20 trials of the functions per

iteration. But the number of iterations could be more elevated than for a Powell optimization, as the search space has a dimension of 7.

GA would be fine for difficult robots, as the tripod, because it allows to encode continuous parameters as well as discrete ones. It however takes a hundred of generations before convergence. So finally it would be too costing in time.

For the third robot, the optimization process will use Powell's algorithm, but with systematical search instead of Golden Section Search Algorithm.

Chapter 7

Conclusion

The main goal of the project was to explore the possibilities of locomotion gaits for the Roombot modules. The three robots presented in this project give a good illustration of the different strategies that can be adopted to find a gait. In addition, a CPG was successfully implemented to the table robot, and different gaits have been compared, showing the best performance for the trot gait, with a maximal speed of 0.83 m/s at maximal torque of 5 Nm. Some mechanical and physical limitation have also been observed in simulation, and permit to understand the behavior of the gait with speed and torque limitations of the motors. It was also demonstrated that Powell's algorithm and Golden Section Search have their limitations, and that physical constraints on the robot has for a consequence to make the fitness landscape uneven.

7.1 Further Work

Sensor feedback could be implemented in the table robot to see if it leads to performance improvements or a better stability. The optimization methods used for this projects result disappointing. Maybe another optimization process would be more efficient. In general, this project has shown that Roombots modules can have interesting various gaits. Next step would be to gather the different aspects of Roombots project, and have a more global strategy about the self reconfigurable pieces of furniture. It would be very interesting to set a strategy for the global locomotion: would it be better to learn gaits by optimization for new structure of the Robot, or to focalize on self-reconfiguration and have only a few robot structures that have efficient gaits?

Bibliography

[1] Y. Bourquin A. Sproewitz, M. Asadpour and A.J. Ijspeert. An active connection mechanism for modular self-reconfigurable robotic systems based on physical latching. In *Proceedings of the 2008 IEEE International Conference on Robotics and Automation (ICRA 2008)*, 2008.

[2] Alberto Behar Andres Castano and Peter Will. The conro modules for reconfigurable robots. In *IEEE/ASME Trans. Mechatronics, 7(4):403-409*, December 2002.

[3] Lipson H. Bongard J., Zykov V. Resilient machines through continuous self-modeling. *Science*, Vol. 314. no. 5802:pp. 1118 – 1121, 2006.

[4] Sbastien Gay. A kinematics-dependent reconfiguration algorithm for chain-type modular robots, 2007.

[5] Houxiang Zhang Jianwei Zhang Juan Gonzalez-Gomez, Eduardo Boemo. Locomotion capabilities of a modular robot with eight pitch-yaw-connecting modules.

[6] Jerome Maye. Control of locomotion in modular robotics, 2007.

[7] S. Murata and H. Kurokawa. Self-reconfigurable robot: Shape-changing cellular robots can exceed conventional robot flexibility. In *Robotics and Automation Magazine*, March 2007.

[8] K. Minami N. Inou and M. Koseki. Group robots forming a mechanical structure-development of slide motion mechanism and estimation of energy consumption of the structural formation. In *Proc. IEEE Int. Symp. Computational Intelligence in Robotics and Automation*, 2003.

[9] L. Righetti and A.J. Ijspeert. Pattern generators with sensory feedback for the control of quadruped locomotion. In *Proceedings of the 2008 IEEE International Conference on Robotics and Automation (ICRA 2008)*, 2008.

[10] John Suh Serguei Vassilvitskii, Mark Yim. A complete, local and parallel reconfiguration algorithm for cube style modular robots. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, 2002.

[11] Mytilinaios E. Desnoyer M. Lipson H. Zykov, V. Evolved and designed self-reproducing modular robotics. In *IEEE Transactions on Robotics, Vol. 23. No. 2, pp. 308 - 319.*, 2007.