ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

BIOLOGICALLY INSPIRED
ROBOTICS GROUP (BIRG)

Control of Locomotion in Modular Robotics

Master Thesis

23 February 2007

Author : Jérôme Maye

Professor : Auke Ijspeert

Supervisor : Alexander Sproewitz

Abstract

This master project focuses on the control of locomotion in modular robotics. We are particularly interested in applying the Central Pattern Generator (CPG) approach to the modular robot YaMoR. The concept of CPG has been introduced in the eighties to explain the mechanism of locomotion in vertebrates. A CPG allows to control multiple antagonist muscles and to modulate the generated pattern with simple high-level stimuli.

In YaMoR, the CPG is modelized as as system of oscillators, whose outputs control the servo motors of the modules. These oscillators contain many free control parameters that have to be tuned for creating a satisfying pattern on the whole mounted robot. As the number of parameters increases, it becomes advantageous to use learning algorithms.

It was shown previously that an algorithm, called Powell's method, gives outstanding results in simulations. Powell's method is a fast and simple heuristic, generally used on mathematical functions, for finding the minimum of a multi-dimensional function.

This project mainly aims at validating the results of the simulations on the real robot platform YaMoR. To prove the efficieny of Powell's method, a comparison with another algorithm is also proposed.

Acknowledgments

First and foremost, I am deeply indebted to my supervisor, Alexander Sproewitz. Without him, the new version of YaMoR would never have been ready in time. Furthermore, he was always available for helping on hardware troubles. Finally, he did a fantastic work on Matlab for displaying the results and nice snapshots of the robot.

Special thanks go to Alessandro Crespi for his kindness and availability. He was of great support in win32 programming.

I also owe a great debt to Rico Moeckel, the developer of the Scatternet protocol. Without his preliminary work, the development of the CPG in YaMoR would have been much more complicated.

A great thanks goes to Professor Auke Jan Ijspeert, the director of the BIRG. He was also always available for answering to questions and allowed me to work on the YaMoR project.

Finally, I have to thank Addamo Maddalena, for the design of the new version of YaMoR.

Table of contents

Chapter 1

Introduction

Self-reconfigurable modular robotics has become an attractive field of research these recent years. In this revolutionary approach, a robot (Figure 1.1) is constructed out of multiple homogeneous building blocks. These modules are connected together at specific docking points in order to create a satisfying shape. The elements typically contain some processing power, sensing facilities, and actuation mechanism for performing the desired task. Furthermore, the robot should be able to dynamically change its mechanical structure, i.e. the way the modules are joint, in response to some sensory inputs.

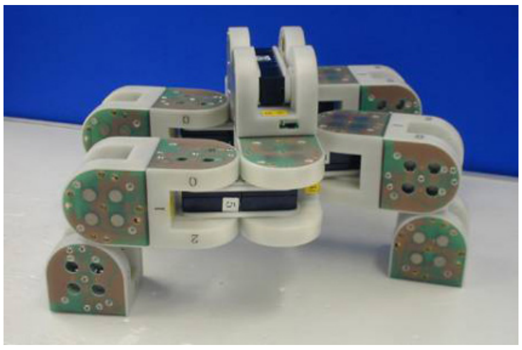


Figure 1.1: Example of a self-reconfigurable modular robot: M-TRAN II (Kurokawa *et al.*, 2003).

In a traditional robot, habitually designed as a specialized and centralized system dedicated to a particular task, the slightest failure most likely leads to a breakdown. In modular robotics, a broken element can be rapidly exchanged at a small cost without disturbing the overall behavior of the system. This process tends to be so simple that some advanced robots are even able to repair themselves or replicate themselves (Figure 1.2). Moreover, the design and manufacture of a single custom robot requires hours of engineering work, while a modular robot is assembled from reusable mass-produced building blocks.

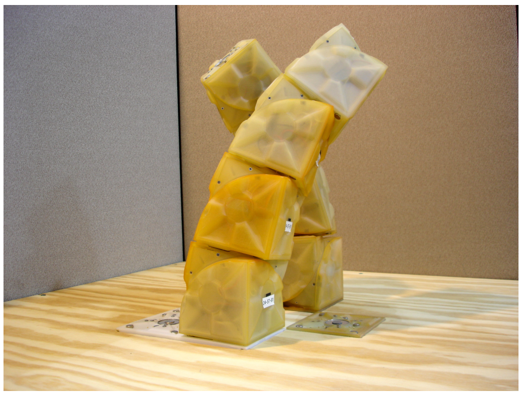


Figure 1.2: Example of a self-replicating robot: Molecube (Zykov *et al.*, 2005).

Although the state of the art do not fulfill all the necessary requirements, an open-ended world of applications is predicted to result from the modular robotics concepts and promises. In space exploration, for instance, modular robotics is of particular interest. Self-repairing and self-replicating machines, able to dynamically change their morphology, would dramatically lower the cost of planet discovery and colonization. Modular robots could also be spread into a devastated environement for search and rescue activities. Amphibious modular robots could participate in deap see exploration and mining. More recently, domestic application is envisioned. Thanks to their low prices, consumers could stock a quantity of those building blocks in a container in their garage. On demand, the modules would assemble into the most appropriate shape for a task like cutting the grass, servicing a car, or cleaning a room. Lastly, the original concept of *intelligent furnitures* has been introduced (Arredondo, 2006). In this project (Figure 1.3), the modules serve as building material for a chair, a table, or a stool. According to the needs of the user, these furnitures could change their shape dynamically and autonomously move to some cover in case of rain for instance.

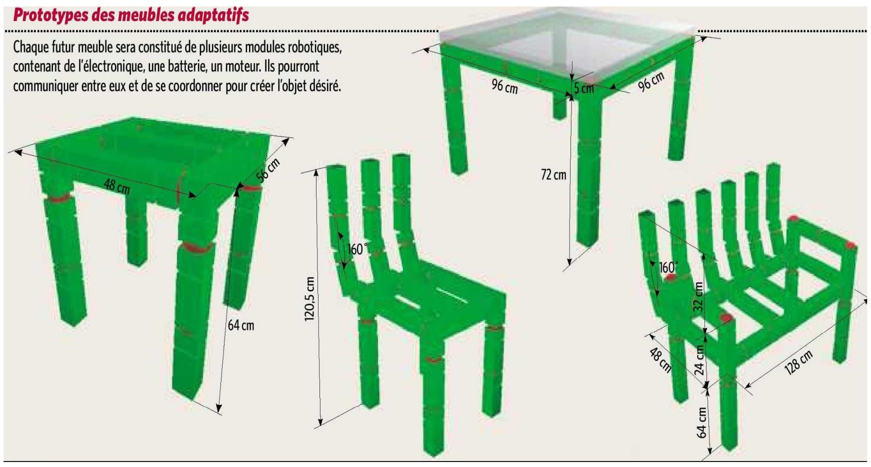

Prototypes des meubles adaptatifs
Chaque futur meuble sera constitué de plusieurs modules robotiques, contenant de l'électronique, une batterie, un moteur. Ils pourront communiquer entre eux et de se coordonner pour créer l'objet désiré.
96 cm
5 cm
96 cm
72 cm
48 cm
56 cm
64 cm
120,5 cm
160°
160°
32 cm
48 cm
24 cm
128 cm
64 cm

Figure 1.3: Example of intelligent furnitures: Roombots (Bloch, 2006).

Among the challenges to be overcome in modular robotics is the control of locomotion. In effect, the majority of the applications suppose the robot is capable of moving efficiently to a certain goal. This necessitates the accurate coordination of multiple degrees of freedom in order to produce an adequate pattern of locomotion, which in turn depends on the environment and on the structure of the robot. Various approaches have been proposed to solve this complex problem ((Kamimura *et al.*, 2004) and (Yim, 1994) for instance). In our project, we focus on the Central Pattern Generator (CPG) biological paradigm. A CPG is a network of neurons (or even a single neuron) which is able to exhibit coordinated rhythmic activity in the absence of any sensory input and is thought to be an essential component of vertebrate locomotion. The transfer of the CPG concept to modular robotic locomotion is detailed in Chapter 2.

In a complex robot, an architecture for the control of locomotion involves a large number of free parameters. An a priori analytical approach for finding the optimal values of those variables is generally inappropriate. For this reason, the parameters are *learnt* with some algorithms, either *online*, i.e. on the robot itself, or *offline*, i.e. on a simulator. In our work, we apply two different online learning algorithms, described in Chapter 2, for letting the robot discover the optimal parameters for its control architecture.

1.1 Project Goals

Since 2004, the Biologically Inspired Robotic Group (BIRG) of the Ecole Polytechnique Federale de Lausanne (EPFL) has started a modular robotic project called YaMoR, standing for *Yet another Modular Robot*. The robot (see Figure 1.4) is constructed out of uniform low-cost modules, equipped with processing power and a servo motor, and able to communicate between each other through a Bluetooth network. More information about the YaMoR project is proposed in Chapter 3.

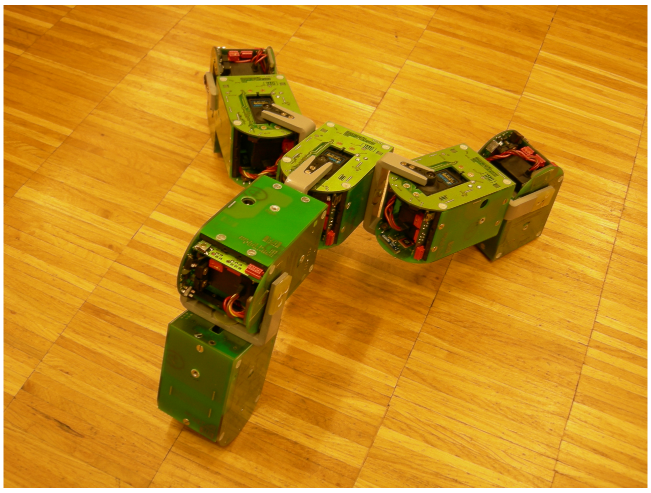


Figure 1.4: Example of a YaMoR structure

In (Marbach and Ijspeert, 2006), Marbach *et al.* show appealing results about online learning of locomotion in a simulated modular robot, using Powell's method (explained in Chapter 2), a fast heuristic for function minimization. In this paper, the parameters of a CPG based control architecture are quickly optimized (approximately 20 minutes) for generating an efficient gait.

The main goal of this master project is to validate the results of Marbach on the newly built YaMoR platform. As can be read in Chapter 3, this firstly requires a way of measuring the displacement of the robot on the floor, the design of a whole firmware for the modules, and of a control application running on a remote PC for the management of the optimization. It then involves long hours of testing and debugging of a distributed system, where the origin of a failure is often arduous to trace. Finally, systematic experiments on various robot shapes are crucial for ensuring the soundess of the results.

This project also aims at providing a clearer understanding of Powell's method and of its efficiency. A comparison, in terms of speed and quality of results, with another optimization algorithm (Particle Swarm Optimization) is proposed as well. Lastly, a simulation environment (under Webots^{TM}) that exactly mimics our system is provided, along with the results of some experiments.

1.2 Outline of the Report

The rest of the report is organized as follows:

Chapter 2 gives the necessary theoretical background for understanding the basis of our method, namely the CPG control architecture for locomotion, and the two optimization algorithms Powell's method and Particle Swarm Optimization.

Chapter 3 describes all the elements involved in the experiments. This includes the YaMoR robot specification, the way the modules communicates via Bluetooth, the fitness evaluation mechanism, the way the CPG is implemented in the modules, the PC control software, and the simulation environment.

Chapter 4 quantitatively details the results of the online learning experiments on three different robot shapes, namely a snake, a triped, and a quadruped robot.

Chapter 5 provides an interpretation of the experimental results and a comparison between the two optimization algorithms.

Chapter 6 concludes this report and gives some future directions of research. Some hints for the hardware, as well as the software, improvement are finally proposed.

Chapter 2

Theoretical Background

This chapter contains the necessary preliminary knowledge for building the experimental setup. The Central Pattern Generator (CPG) control architecture, which lies at the basis of the YaMoR locomotion, is firstly introduced. The Powell's optimization method, used in (Marbach and Ijspeert, 2006), is then clearly detailed. Finally, for comparison purpose, another optimization algorithm is explained, namely Particle Swarm Optimization. As can be read in Chapter 3, the CPG is implemented in a distributed manner in YaMoR. The optimization algorithms, running on a remote PC, have to fix the free parameters of the CPG such that an efficient forward locomotion is generated.

2.1 CPG Model

In the sixties, experiments on a decerebrated cat (Shik *et al.*, 1966) have shown that the locomotion mechanism in vertebrates is entirely located in the spinal cord and that high-level stimulus from the brain only modulate the kind of gait observed (walk, trot, rest, etc.). In (Delcomyn, 1980), this mechanism is referred to as a CPG, a network of neurons able to generate coordinated rhythmic patterns, even in absence of any sensory input. The output of the CPG directly or indirectly controls the activity of the antagonist muscles used for locomotion.

As in vertebrates, modular robots generally contain multiple actuated points that have to be synchronized for generating locomotion. It is thus tempting to reproduce the CPG behavior, i.e. a robust distributed system, in modular robotics. Professor Ijspeert[1], active in the CPG modelization for several years (Ijspeert and Kodjabachian, 1999), has tailored an interesting model for YaMoR. It will be thoroughly explained in the rest of the section.

[1]`auke.ijspeert@epfl.ch`

2.1.1 Mathematical Equations

Professor Ijspeert models the CPG as systems of coupled nonlinear oscillators. An oscillator i is implemented as follows:

$$\dot{\phi}_i = \omega_i + \sum_j (\omega_{ij} r_j sin(\phi_j - \phi_i - \varphi_{ij}) \tag{2.1a}$$

$$\ddot{r}_i = a_r(\frac{a_r}{4}(R_i - r_i) - \dot{r}_i) \tag{2.1b}$$

$$\ddot{x}_i = a_x(\frac{a_x}{4}(X_i - x_i) - \dot{x}_i) \tag{2.1c}$$

$$\theta_i = x_i + r_i cos(\phi_i) \tag{2.1d}$$

where θ_i is the oscillating set-point (in radians); ϕ_i, r_i and x_i are state variables corresponding to the phase, the amplitude, and the offset (in radians); ω_i, R_i, X_i are control parameters for the desired frequency, amplitude, and offset (in radians); ω_{ij} and φ_{ij} are coupling weights and phase biases which determines the relation between oscillator i and j; r_j and ϕ_j are received from neighbor j; a_r and a_x are constant positive gains ($a_r = a_x = 4$[rad/s]) that control the speed of convergence of r_i and x_i.

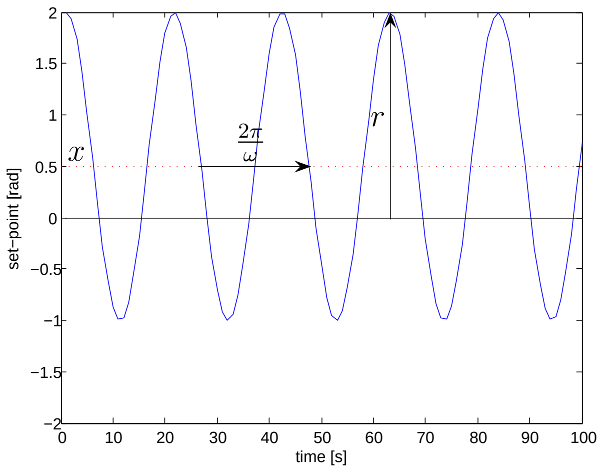


Figure 2.1: Example of an oscillator output

2.1.2 Convergence

The above equations ensure that the output of the oscillator θ_i exhibits limit cycle behavior, i.e. produces a stable periodic output. Equation 2.1a reflects the time evolution of the oscillators. In this work, the same frequency $\omega_i = \omega$ is shared by all the oscillators. The coupling between oscillators i and j is such that the phases sum up to zero ($\varphi_{ij} = -\varphi_{ji}$) (see Figure 2.2). Furthermore, in the case of a closed loop of oscillators, all the phase biases in the loop sum up to a multiple of 2π (see Figure 2.2). Taking this into account, Equation 2.1a ensures that the phases converge to a regime in which they grow linearly at the common rate ω.

Equations 2.1b and 2.1c are second order linear equations which have respectively R_i and X_i as stable fixed points. Whenever R_i and X_i are changed, the state variables r_i and x_i will asymptotically and monotonically converge to R_i and X_i. This means that we can smoothly modulate the amplitude and the offset of the oscillators.

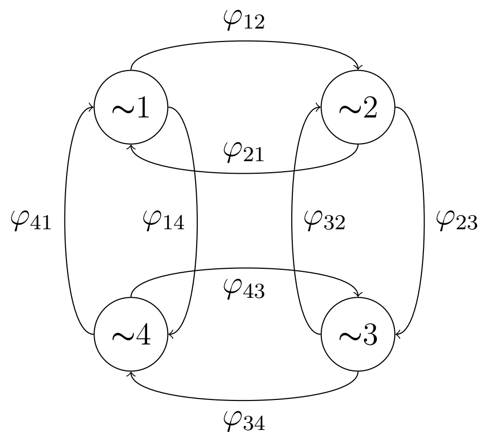


Figure 2.2: Example of a valid CPG architecture. $\varphi_{12} = -\varphi_{21}$, $\varphi_{23} = -\varphi_{32}$, $\varphi_{34} = \ \varphi_{43}$, $\varphi_{41} = -\varphi_{14}$, $\varphi_{12} + \varphi_{23} + \varphi_{34} + \varphi_{41} = 2\pi n$ and $n \in \mathbb{N}$

Given the above settings, two oscillators i and j coupled together with non-zero weights ω_{ij} asymptotically converge to limit cycles $\theta_i^\infty(t)$ and $\theta_j^\infty(t)$ defined by the following closed form solutions:

$$\theta_i^\infty(t) = X_i + R_i cos(\omega t + \varphi_{ij} + \phi_0) \tag{2.2a}$$
$$\theta_j^\infty(t) = X_j + R_j cos(\omega t + \varphi_{ji} + \phi_0) \tag{2.2b}$$

where ϕ_0 depends on the initial conditions of the system. Due to the common frequency and the consistency of phase lags in a loop, the coupling weights only affect the speed of convergence to the limit cycle (higher weights mean faster convergence).

From 2.2a and 2.2b, we can derive that the system stabilizes into phase-locked oscillations for the oscillators connected together. The oscillations are then modulated by the control parameters, namely ω for setting the common frequency, ϕ_{ij} for setting the phase lags between two connected oscillators i and j, R_i for setting the amplitude of oscillator i, X_i for setting the offset of oscillator i, and ω_{ij} for setting the coupling weight between two connected oscillators. The oscillators quickly returns to a limit cycle after any perturbation, as we will see in an example in Chapter 3, where we show the actual implementation.

2.1.3 Properties

The above CPG model is interesting for designing locomotion controllers and doing online optimization for several reasons. First of all, the system has a limit cycle behavior, i.e. it returns rapidly to a steady-state after any change of the state variables. Secondly, the limit cycle of the system has a closed form solution[2], which is cos-based and has explicit control parameters (ω, R_i, and X_i). We can thus easily influence the oscillators with desired relevant features such as the frequency, the amplitude, and the offset. Lastly, the control parameters can be abruptly and/or continuously changed without damaging the motors, because they induce only smooth modulations of the set-point. Since optimization algorithms have to explore the whole parameter space, this is a critical property in the case of online learning.

[2]It solves a given problem in terms of functions and mathematical operations from a given generally accepted set.

2.2 Optimization Algorithms

Given a function $f(\vec{x})$, the task of an optimization method is to find the vector $\vec{x}$ such that f achieves a minimum or maximum value. Since minimizing a function f is the dual problem of maximizing the function $-f$, the rest of the chapter will focus on minimization.

An optimum of a function f can be *global* (valid in the whole domain of f) or *local* (valid in some finite neighborhood of f). Some algorithms are more prone to fall into local optima, while others systematically reach the global optimum, owing to more exploration. A tradeoff between accurate optimum and computation time is often an issue.

To our great disappointment, a perfect optimization algorithm does not exist for a particular application. A variety of choices, that enhance certain desired characteristics, are rather encountered. In the case of online learning on a real robot platform, the speed of convergence to a fairly good solution, without too many evaluations of the fitness function, is an essential criterion. Furthermore, the algorithm should be able to optimize the function without gradient information, since the cost of its computation is not affordable. In the rest of the section, two optimization algorithms, that meet the above criteria, are presented in detail.

2.2.1 Powell's Method

Powell's method is a fast heuristic for finding the minimum of a multivariable function $f(\vec{x})$, such that $\vec{x} = [x_1 x_2 ... x_N]^T$ and N is the dimension. One iteration of Powell consists of N one-dimensional line minimizations, with respect to a direction set updated according to the shape of the function. Since Powell's method is not able to find several minima, we assume that f has one global optimum and the case of local optima will be explored in Chapter 4.

One-Dimensional Minimization

One-dimensional minimization is performed with Brent's method (Algorithm 1). Given an initially bracketed minimum, this heuristic uses a combination of golden section search and parabolic interpolation for reaching the minimum of the function. The initial bracketing process is described in Chapter 3.

The golden section search iteratively manipulates a bracketing triplet of points (a, x, b), such that $a < x < b$ (or $b < x < a$), $f(x) < f(a)$ and $f(x) < f(b)$. The point u that is a fraction $\gamma = 0.38197$[3] into the larger of the two intervals (starting from x) is first evaluated. For instance, imagine $x \geq \frac{b-a}{2}$ and thus u is chosen in $[a, x]$. If $f(u) > f(x)$, then the next bracketing triplet will be $(a = u, x, b)$; otherwise if $f(u) < f(x)$, a new minimum has been found and the bracketing triplet is updated as $(a, x = u, b = x)$. This process continues until the distance between a and b is conveniently small. At each iteration, the size of the interval is reduced by a factor $\psi = 0.61803$ ($\psi = 1 - \gamma$), it is thus said to converge linearly. If ϵ_n is the size of the interval at iteration n, then $\epsilon_{n+1} = \psi * \epsilon_n$. As a picture is worth a thousand words, Figure 2.3 shows an example of the golden section search.

[3]The interval always respects the famous *golden ratio* that has been widely used in architecture and art. It can be mathematically proved (Press *et al.*, 1988) that this is the optimal way of reducing the interval.

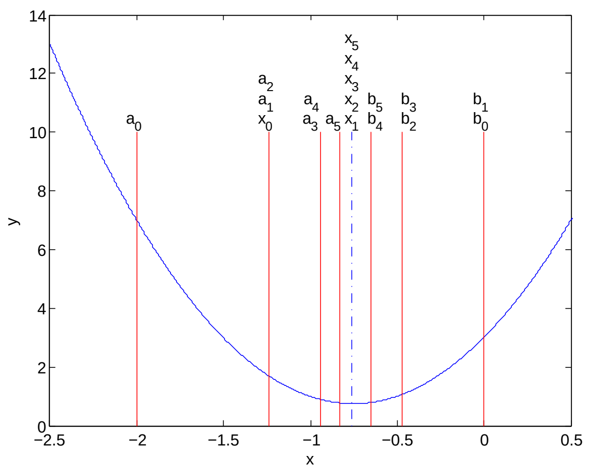


Figure 2.3: Example of golden section search. $f(x) = y = 4x^2 + 6x + 3$ is the function under minimization. An analytical method finds the exact minimum at $x = -0.75$ (root of $f'(x)$). The golden section search is started with the interval $[a_0 = -2.00, x_0 = -1.23, b_0 = 0.00]$, which respects the preconditions. The interval are then iteratively updated as $[a_1 = -1.23, x_1 = -0.76, b_1 = 0.0]$, $[a_2 = -1.23, x_2 = -0.76, b_2 = -0.47]$, $[a_3 = -0.94, x_3 = -0.76, b_3 = -0.47]$, $[a_4 = -0.94, x_4 = -0.76, b_4 = -0.65]$, $[a_5 = -0.83, x_5 = -0.76, b_5 = -0.65]$. After 5 iterations, the interval has a size of 0.18 and the minimum is -0.76. Note that the final size of 0.18 is equal to the original size of 2.00 times ψ^5.

The golden section search never fails to reach the minimum. However, in some cases, when the function is nicely parabolic, a faster method for getting to it could be used. The idea is to remember the two previous values of the minimum x, let say v and w. A parabolic fit is then drawn through the three points x, v, w. In the next step, the minimum of this parabola is reached, let say u, and the function is evaluated at this point. The new bracketing triplet becomes $(a, x = u, b)$ if $f(u) < f(x)$, otherwise u replaces one of the limit a or b. The process continues until the interval is tolerably small. The parabolic interpolation method converges *superlinearly*. If ϵ_n is the size of the interval at iteration n, then $\epsilon_{n+1} = constant * (\epsilon_n)^m$ with $m > 1$. Figure 2.4 shows an example of the parabolic interpolation.

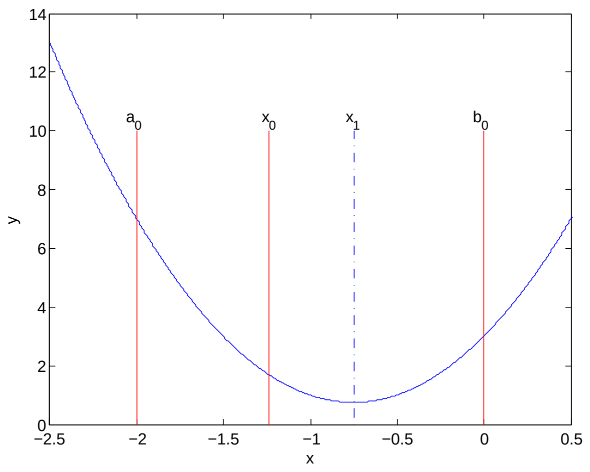


Figure 2.4: Example of parabolic interpolation. The function to minimize is still $f(x) = y = 4x^2 + 6x + 3$. In one step, the minimum $x_1 = -0.75$ is reached. This result corresponds to the exact one, but was expected since f is exactly a quadratic parabola.

Algorithm 1 Brent's Method (simplified)

```
Require: $a$, $x$, $b$ such that $(a < x < b$ or $b < x < a)$ and $(f(x) < f(a)$ and $f(x) < f(b))$
 1: variables renaming s.t. bracketing interval is $(a, x, b)$ and $a < x < b$
 2: $w \Leftarrow x$, $v \Leftarrow x$
 3: repeat
 4:     $u \Leftarrow x - \frac{1}{2}\frac{(x-v)^2(f(x)-f(w))-(x-w)^2(f(x)-f(v))}{(x-v)(f(x)-f(w))-(x-w)(f(x)-f(v))}$
        {formula for the minimum of the parabola fitting $x, v, w$, using $f'(x) = 0$ and solving a linear system of equations}
 5:     if $u \notin [a, b]$ or step do not enough reduce the interval then
 6:         $u \Leftarrow x - \gamma * (a - x)$
            {golden section, case where $[a, x]$ is the larger of the two sub-intervals}
 7:     end if
 8:     if $f(u) < f(x)$ then
 9:         $w \Leftarrow v$, $v \Leftarrow x$, $b \Leftarrow x$, $x \Leftarrow u$ {case $u \in [a, x]$}
10:     else
11:         $a \Leftarrow x$ {case $u \in [a, x]$}
12:     end if
13: until $b - a \leq \epsilon$ {discussed later}
```

Multi-Dimensional Minimization

Starting from an initial guess of the minimum $\vec{x}_0$, the next approximation $\vec{x}_1$ can be estimated by iteratively projecting f on one dimension and minimizing the resulting single variable function with Brent's method. This method generates the sequence of points $\vec{x}_0 = \vec{p}_0$, $\vec{p}_1$, $\vec{p}_2$, ..., $\vec{p}_N = \vec{x}_1$, where $\vec{p}_i$ is the minimum of f along the direction given by the i^{th} standard base vector. This method is appealing, but can be highly inefficient in some cases (see Figure 2.5). Powell improves it dramatically (Algorithm 2).

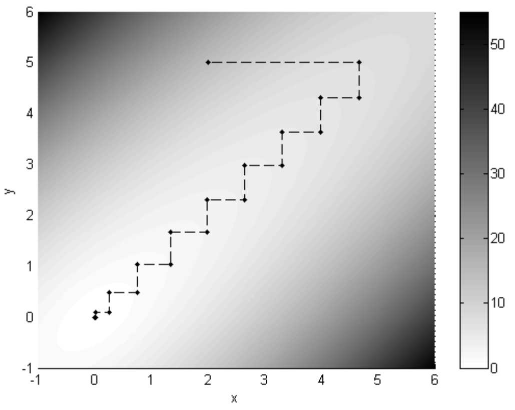


Figure 2.5: Example of a function where naive method is inefficient, because of too small steps.

The point $\vec{x}_1$ can be viewed as the minimum of f along the direction $\vec{p}_N - \vec{p}_0$. As it seems to be a good direction, it could be included in our direction set for the next iteration under some conditions. A point is extrapolated further in this direction to see if the function continues to decrease. The best choice for avoiding that the direction set becomes linearly dependent (and thus $\{\vec{x}\}_{k=0}^{\infty}$ do not converge to the minimum) is that the direction $\vec{p}_N - \vec{p}_0$ replaces the direction N and this latter the direction along which f had its largest decrease. It is replaced only if it was a major part of the total decrease in f. The stopping criteria of Powell's method will be discussed in Chapter 3. A complete example of Powell's method is detailed in Figure 2.6.

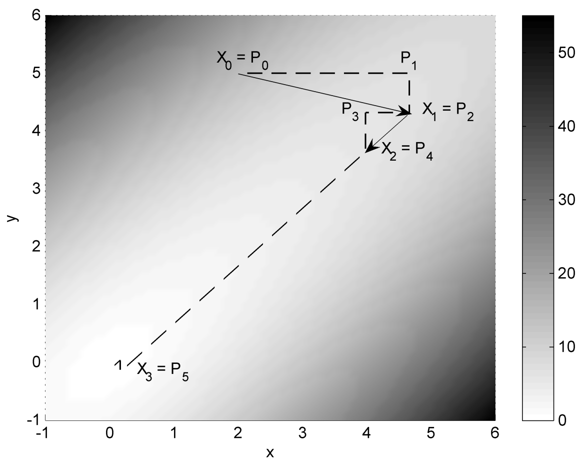


Figure 2.6: Example of Powell's method. The function to minimize is $f(x, y) = z = \sqrt{x^2 + y^2} + (x - y)^2$. The algorithm is started from the first guess of the minimum, the point $X_0 = [2\ 5]^T$. f is minimized with Brent along the standard unit vector $\vec{e}_1 = [1\ 0]^T$ from $X_0 = P_0$ up to $P_1 = [4.66\ 5]^T$. Along $\vec{e}_2 = [0\ 1]^T$, the point $P_2 = [4.66\ 4.32]^T$ is reached, which is the new minimum X_1. The point extrapolated in the direction $P_2 - P_0$ does not reduce f, so the direction set is kept. The points $P_3 = [3.98\ 4.32]^T$ and $P_4 = [3.98\ 3.64]^T = X_2$ are found in the next iteration. Since the extrapolated point is lower, f is minimized along $X_2 - X_1$ to the point $P_5 = [0.16\ 0.16]^T = X_3$. For the next iteration, the direction set is updated to $[0\ 1]^T$ and $X_2 - X_1$.

Algorithm 2 Powell's Method

```
Require: $i = 0$, $\vec{u} = [\vec{u}_1\vec{u}_2...\vec{u}_N] = [\vec{e}_1\vec{e}_2\vec{e}_N]$ with $\vec{e}_i$ standard base vector of dimension $i$, $\vec{x}_0$ an initial guess
Ensure: $\{\vec{x}\}_{k=0}^{\infty}$ converges to the minimum of $f$
 1: repeat
 2:   $\vec{p}_0 \Leftarrow \vec{x}_i$
 3:   for $k = 1$ to $N$ do
 4:     find the value of $\gamma_k$ that minimizes $f(\vec{p}_{k-1} + \gamma_k\vec{u}_k)$ using Brent's method
 5:   end for
 6:   $r$ equals the maximum decrease of $f$ along one direction, $\vec{u}_r$ equals this direction
 7:   $i \Leftarrow i + 1$
 8:   if $f(2\vec{p}_N - \vec{p}_0) < f(\vec{p}_0)$ and
      $2(f(\vec{p}_0) - 2f(\vec{p}_N) + f(2\vec{p}_N - \vec{p}_0))(f(\vec{p}_0) - f(\vec{p}_N) - r)^2 < r(f(\vec{p}_0) - f(2\vec{p}_N - \vec{p}_0))^2$ then
 9:     $\vec{u}_r \Leftarrow \vec{u}_N$, $\vec{u}_N \Leftarrow \vec{p}_N - \vec{p}_0$ {discard direction with biggest decrease}
10:     find the value of $\gamma$ that minimizes $f(\vec{p}_0 + \gamma\vec{u}_r)$ using Brent's method
11:     $\vec{x}_i \Leftarrow \vec{p}_0 + \gamma\vec{u}_r$
12:   else
13:     $\vec{x}_i \Leftarrow \vec{p}_N$
14:   end if
15: until $f(\vec{x}_{i-1}) - f(\vec{x}_i) \leq \epsilon$ {discussed later}
16: return $\vec{x}_i$
```

2.2.2 Particle Swarm Optimization

Particle Swarm Optimization (PSO) (Kennedy and Eberhart, 1995) is a rather revolutionary stochastic optimization technique, inspired by the movement of flocking birds and their interactions with their neighbors. If one of the birds detects a good spot for food, the rest of the swarm is able to approach this point quickly, even the individuals at the opposite side of it. Furthermore, in order to favor the exploration of the entire search space, each bird has a certain degree of randomness in its movement.

Algorithm

PSO (Algorithm 3) models the set of potential problem solutions as a swarm of particles moving in a virtual search space. A swarm of particles is deployed in a virtual search space with random initial positions $x_{i,j}$ and velocities $v_{i,j}$, where i represents the index of the particle and j the dimension in the search space. Each particle flies through the virtual space, attracted by positions that yielded the best performances. The position at which a particle achieved its best performance is recorded as $x^*_{i,j}$. Each particle keeps also track of the performance of the best individual in its neighborhood, recorded as $x^*_{i',j}$. The neighborhood of a particle can be global, i.e. it keeps track of the entire swarm. In this case, exploitation is favored and the algorithm is more likely to fall into a local optimum. On the other hand, a particle can have a local neighborhood, i.e. it keeps track of a subset of the swarm. In this case, the convergence is slower, exploration is favored and it is thus less likely to fall into a local optimum. Therefore, there is a tradeoff between speed of convergence and avoiding to fall into local optima. In our experiments, the local neighborhood of a particle i is composed of the elements $i-1$ and $i+1$, modulo the size of the swarm. At each iteration, the algorithm updates the variables as follows:

$$v_{i,j} = w * (v_{i,j} + pw * rand() * (x^*_{i,j} - x_{i,j}) + nw * rand() * (x^*_{i',j} - x_{i,j})) \tag{2.3a}$$

$$x_{i,j} = x_{i,j} + v_{i,j} \tag{2.3b}$$

where w is an inertia coefficient, pw and nw are the weights of attraction to the previous local best performance of the particle and of its neighborhood respectively, and $rand()$ is a uniformly-distributed random number in [0,1]. The value of the parameters will be discussed in Chapter 4. The stopping criteria is based on the number of iterations. An example of PSO is depicted on Figure 2.7.

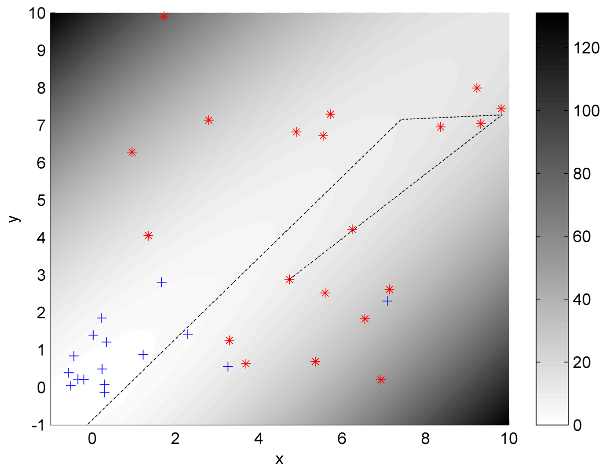


Figure 2.7: Example of Particle Swarm Optimization. The function to minimize is $f(x,y) = z = \sqrt{x^2+y^2} + (x-y)^2$. 20 particles (red stars) are spreaded randomly in a limited range of the space. After 15 iterations of PSO, although most of the particles (blue crosses) are situated near the minimum, there are still some distant individuals that explore the space. The dash-dot black line shows a part of the path of a particle.

Algorithm 3 Particle Swarm Optimization

```
Require: $N$ = number of dimensions, $M$ = number of particles, $T$ = number of iterations
 1: for $i = 1$ to $M$ do
 2:    for $j = 1$ to $N$ do
 3:       init $x_{i,j}$, $v_{i,j}$ randomly
 4:    end for
 5: end for
 6: $t \Leftarrow 0$
 7: repeat
 8:    for $i = 1$ to $M$ do
 9:       compute fitness of particle $i$
10:    end for
11:    for $i = 1$ to $M$ do
12:       update $lbest_i$ {$lbest_i$ is the local best performance of $i$}
13:       update $nbbest_i$ {$nbbest_i$ is the neighborhood best performance of $i$}
14:    end for
15:    for $i = 1$ to $M$ do
16:       for $j = 1$ to $N$ do
17:          update $v_{i,j}$ {Equation 2.3a}
18:          update $x_{i,j}$ {Equation 2.3b}
19:       end for
20:    end for
21:    $t \Leftarrow t + 1$
22: until $t = T$
23: return $max(lbest)$
```

Chapter 3

Experimental Setup

In this chapter, the major part of our work for this master project is utterly presented. This encompasses the development of a Light Emitting Diode (LED) tracking algorithm, a central control program for the management of the robot and of the optimization algorithms, and a firmware, that implements the Central Pattern Generator (CPG) model, for the microcontrollers of the modules. The modular robot hardware and the Bluetooth® communication protocol are introduced as a preliminary to our work.

3.1 YaMoR Project

YaMoR, standing for *Yet another Modular Robot*, is a modular robotics project initiated and developed at the Biologically Inspired Robotics Group (BIRG) of the Ecole Polytechnique Fédérale de Lausanne (EPFL). YaMoR consists of several homogeneous modules that can be connected together. A first promising version (Figure 3.1, left) is built during the summer semester 2004 and a Java software (Bluemove) allows to experiment different kind of locomotion controllers on several robot shapes (Moeckel *et al.*, 2006).

During the winter semester 2005, based on the previoulsy acquired knowledge, a new version is designed by a master student. The first working module (Figure 3.1, right) is mounted in the summer 2006, thanks to the endeavours of Alexander Sproewitz[1], who seriously improved the system. In this version, one module weights 0.25 kg and has a length of 94 mm (lever included), with a cross section of 45x50 mm. Each module has a U-shaped lever, with one degree of freedom, that can be moved in a range of 180° approximately. The lever is driven by

[1]PhD student at the BIRG, `alexander.sproewitz@epfl.ch`

a powerful RC[2]-servo, with a maximum rotation speed of 60°/0.16 s and a maximum torque of 73 Nm, sufficient for a module to lift three others. In order to place a lot of electronic components into a small module, Printed Circuits Boards (PCB) serve as outer casing on one side, and as electronic support on the other side. Modules can be statically attached together at specific points. The connection mechanism, based on a screw and a pin, allows fixations with angle at every 15 degree. A YaMoR module (see Figure 3.2 for components) is powered by an onboard Li-Ion battery and contains 7 different boards:

1. a Bluetooth board
2. a board with an ARM microcontroller
3. a board with a Field Programmable Gate Array (FPGA)
4. a sensor board
5. a power board
6. a battery support board for the plus
7. a battery support board for the minus

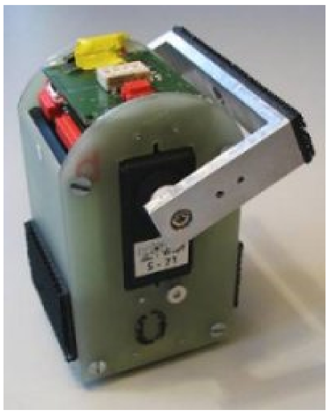
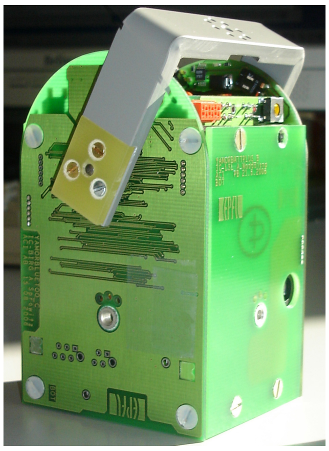



Figure 3.1: First version of YaMoR (left), new version (right)

The Bluetooth board belongs to the external structure of the module and contains a Zeevo chip ZV4002 (Zeevo, 2004) for the wireless communication. The original software delivered with the ZV4002 microcontroller allows the establishment of a point-to-point link between two Bluetooth compliant devices and implements the Serial Port Profile (SPP). This latter acts as a serial line cable replacement (see (Bray and Sturman, 2000) for details), i.e. an input serial stream into the ZV4002 is directly sent via Bluetooth and a Bluetooth packet received from the ZV4002 is forwarded through its output serial port. Nonetheless, the ZV4002 firmware lacks the ability to operate into a network of devices, as the one required by the CPG model of Chapter 2. For this reason, Rico Moeckel[3] has developed a network protocol (Section 3.2) on top of the original firmware.

[2]Typically employed in radio-controlled application.
[3]Ex-internship student at the BIRG and now PhD student in Zurich, `moeckel@ini.phys.ethz.ch`

The microcontroller board is an helper board that can be plugged into the module for extending its processing power. It contains a Philips LPC2138 chip (Philips, 2005), based on an ARM7TDMI^{TM}(Segars *et al.*, 1995) processor architecture. The key features of the LPC2138 are small power consumption, small size, 32 kB of on-chip SRAM, 512 kB of on-chip Flash, 16 10-bit Analogic-to-digital converter channel, one 10-bit Digital-to-analogic converter channel, and a wide choice of peripherals (2 UART, 2 SPI, 2 I^2C, ...).

The FPGA board, pluggable as the microcontroller board, contains a Spartan^{TM}-3 FPGA from Xilinx® (Xilinx, 1999). Although this board is not used in this project, its power could be exploited properly in the future. For instance, the microcontroller board could be totally removed and the system then be directly implemented in hardware, in case a lot of computation power is needed. A combination of softcore processor[4] running C code and hardware implementation for critical functions could as well be used. There is also the possibility to use both the ARM board and the FPGA board in parallel. In this case, the communication between the boards would rely on UART and the system lose its attractivity. A final matter is the power consumption, it is worth keeping in mind that an FPGA consumes more than a microcontroller.

The sensor board lies at the bottom of the outer structure of the module. It contains a PIC®16F876A (Microchip, 2003) from Microchip that processes informations from an infra-red sensor (distance sensor) and a 3D accelerometer. This board is not used in this project, but could in the future serves for the fitness computation and/or for the dynamic attachment of the modules.

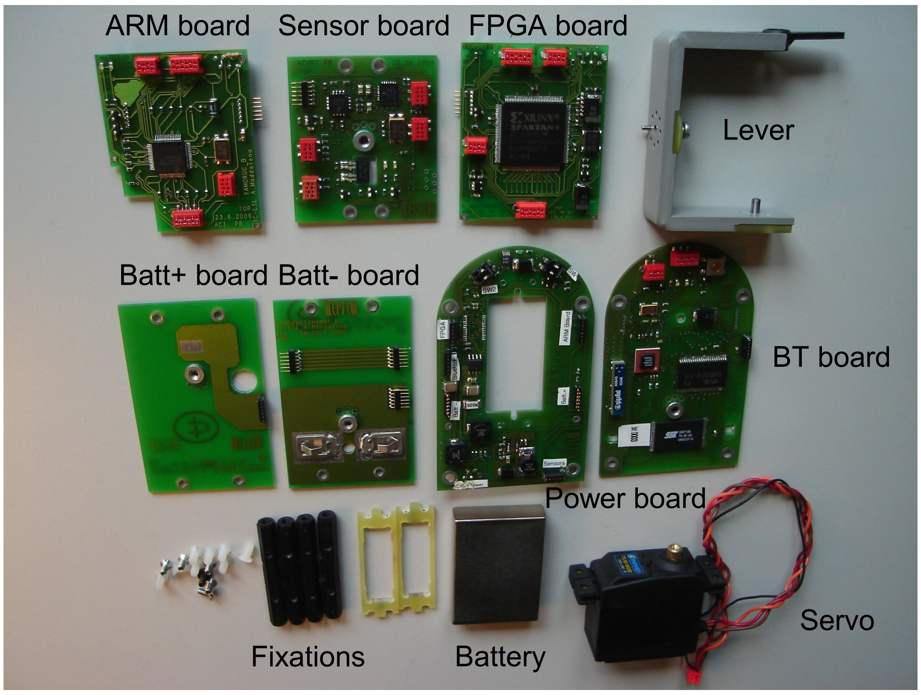


Figure 3.2: Components of a YaMoR module.

[4]MicroBlaze^{TM}soft core processor for instance (Xilinx, 2004).

3.2 Wireless Network

The locomotion control architecture that is used for doing online optimization is based on the CPG model presented in Chapter 2. Each module runs an oscillator that controls the position of the lever. The CPG model assumes each oscillator, i.e. each module, repeatedly sends/receives the amplitude and the phase to/from the neighbors it is coupled. Moreover, it is desirable that a remote PC is linked to the modules. Therefore, a communication network is required for building the CPG architecture.

In modular robotics, especially when dynamic self-reconfiguration is the final aim, the availability of a wireless network, either based on radio or light, is crucial. In YaMoR, Bluetooth (Bray and Sturman, 2000) has been chosen as the wireless communication protocol. This radio technology offers low power consumption (compared to Wireless Local Area Network 802.11 from IEEE for instance (Crow *et al.*, 1997)). Thanks to an ingenious Frequency Hopping protocol, Bluetooth devices can operate in noisy environments (mixed with other radio protocol sharing the same frequency[5]). Since Bluetooth is a standard, every certified devices can communicate together without problems. Finally, the cost of Bluetooth devices has been largely reduced by mass production.

Since Bluetooth was originally designed as a cable replacement system, there are some limitations in the way the devices are to be connected. In its simplest version, the protocol allows a single master (a Bluetooth dongle on a PC for instance) to be connected with up to 7 slaves (keyboard, mouse, ...). This is called a *Piconet* (Figure 3.3, left) in the Bluetooth jargon. In this configuration, the slaves are not supposed to communicate together and thus are not aware of each other. All the traffic from slave to slave passes through the master node. An enhanced version of the protocol allows to build a *Scatternet* (Figure 3.3, right). In the Scatternet, a device can play the role of the master in a Piconet and of the slave in another Piconet. Nonetheless, the unpaired devices are not able to communicate directly and do not know about each other.

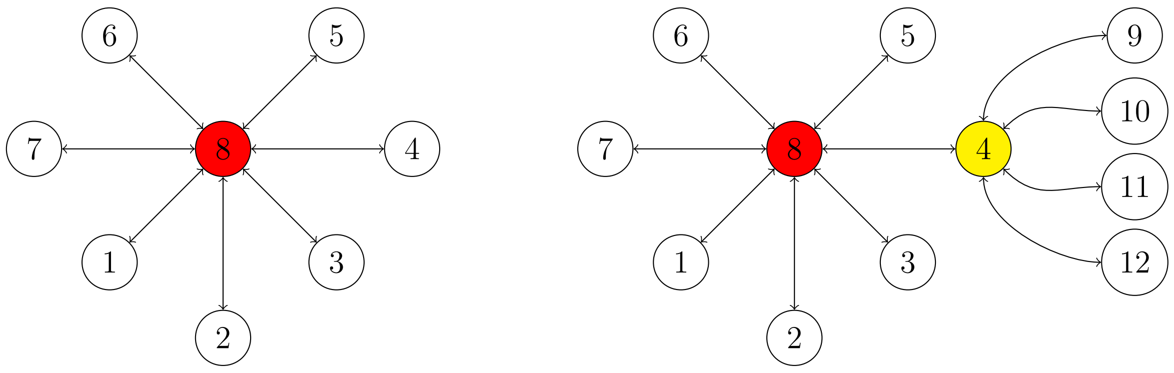


Figure 3.3: Example of Bluetooth Piconet (left), and of Scatternet (right). In the Scatternet, device 4 is a slave in one Piconet, and a master in a second Piconet.

To overcome the difficulty of managing such Bluetooth networks, Rico Moeckel devised a new Scatternet protocol, called SNP, above Bluetooth. SNP makes Bluetooth communication easier, since any device that wants to send data to another device in the network simply launches a packet containing its address in the correct field. SNP is then responsible to bring

[5]Bluetooth uses the license-free ISM band at 2.5 GHz.

the packet to the appropriate receiver, taking the shortest path, regardless of the Bluetooth connection structure. Moreover, SNP is a dynamic pathfinder, i.e. it can adapt when the network structure changes. Finally, as broadcast is also supported, a central host can remotely control all the connected devices.

The SNP protocol has been implemented in the ZV4002 of the Bluetooth board on top of the original firmware. As the SPP is still active, data to send to another device is inputted in the serial port of the chip and data received is also ouputted from this port. Thanks to a filtering part added in the firmware, the data stream is now analyzed with respect to the SNP communication protocol. If a packet is not destinated to the device, it is forwarded in the network. Otherwise, it is ouputted on the serial port. The routing is based on a constantly updated friend table in each device.

The SNP protocol has been first used in this master project and some unexpected limitations have been observed:

- For the initial connections of the devices, a serie of connection packets have to be sent in the network. A minimum delay between these packets has to be respected. Although this delay was empirically fixed at 7 seconds to be conservative, there is a way of lowering it (see Section 3.5).
- It seems that only a chain-like structure (see Figure 3.4) of Bluetooth connections remains stable, otherwise too much traffic comes into the same node simultaneously and causes the ZV4002 reset. This burden may be removed by either introducing an unified time in the network, or use a ask/respond protocol.
- When a serie of data packets have to be sent to different devices from the same point, a minimum delay between packets has also to be respected. It was empirically fixed at 1 second.
- Using broadcast address to send to all the devices in the network is not always reliable. This is probably due to the fact that broadcast is implemented as consecutive sending of data packets to each device in the friend table and the above mentioned delay is not respected.

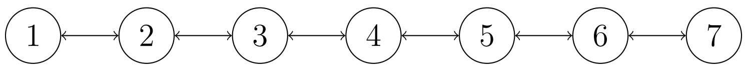


Figure 3.4: Example of a stable Bluetooth connection structure.

3.3 Tracking Algorithm

An optimization algorithm needs a way to estimate how the robot is performing with a given set of parameters. This fitness is computed as a function of the distance moved by a LED attached to one of the module during a given time window. The LED is tracked by a camera connected to a PC which send its position via TCP to the optimization algorithm. The complete setup of the experiment is shown in Figure 3.5.

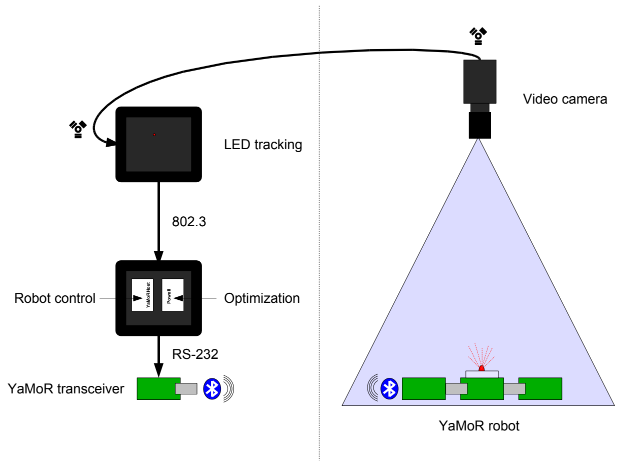


Figure 3.5: Experimental setup

The tracking is done as in Algorithm 4. There is a first filtering of the color image, so as to get a black and white image. The red pixels of the original image that go over a given threshold (200 in our case) are represented as white pixels in the filtered image (white pixels have the value 1 and black pixels the value 0). The idea is then to move a rectangular window of size $WINDOW\_HEIGHT * WINDOW\_WIDTH$ ($WINDOW\_HEIGHT = WINDOW\_WIDTH = 3$) from the top left pixel to the bottom right one and to count the number of white pixels inside it. The rectangle that has the biggest concentration of white pixels is taken as the position of the LED.

The tracking algorithm is quite robust and efficient. It can provide the position of the LED each 39 ms in average. In order to minimize the tracking errors, the experiments are performed in an environment that tends to reduce light reflections. As there is a non negligible distorsion on the camera, a correcting function is implemented on the receiving side of the LED position, to dedicate the processor only to the LED tracking and thus have a higher frequency in the measurements.

The function that corrects the distorsion of the lens was provided as a Matlab® file by Alessandro Crespi[6]. Given a pixel of the corrected image, it computes which pixel from the original image should be placed at this position. The function `xycorr` is as follows:

[6]PhD student at the BIRG, `alessandro.crespi@epfl.ch`

```
static int xycorr(int* x_src, int* y_src, int x_dest, int y_dest,
                  double* params)
{
  double rd, rs, f, scale;
  int    iter;

  rd   = sqrt((x_dest * x_dest) + (y_dest * y_dest)) / params[4];
  rs   = rd;
  f    = (rs * params[0]) + (rs * rs * params[1]) +
         (rs * rs * rs * params[2]) + (rs * rs * rs * rs * params[3]);
  iter = 0;

  while ((fabs(f - rd) > 1e-6) && (iter < 100))
  {
    iter++;
    rs -= (f - rd) / (((4 * params[3] * rs + 3 * params[2]) * rs
           + 2 * params[1]) * rs + params[0]);
    f   = (rs * params[0]) + (rs * rs * params[1]) +
          (rs * rs * rs * params[2]) + (rs * rs * rs * rs * params[3]);
  }

  scale = rs / rd;

  *x_src = (int)(x_dest * scale);
  *y_src = (int)(y_dest * scale);

  return 0;
}
```

An application (`CamAppCorr`) has been written for tuning the parameters of the function. The parameters empirically found are `params={1-0.070, 0, 0.070, 0, IMAGE_WIDTH / 2}`. In the final implementation, the function `xycorr` works from the original pixels to the destination pixels, the parameters becomes then `params={1 + 0.070, 0, -0.070, 0, IMAGE_WIDTH / 2}`. Figure 3.6 shows the correction filter applied to the image. Figure 3.7 shows an example of correction on a real image.

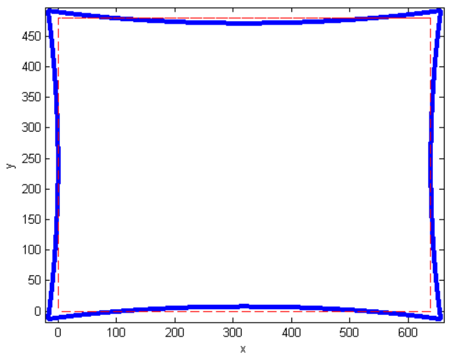


Figure 3.6: Correction filter (blue) applied to the original image (red)

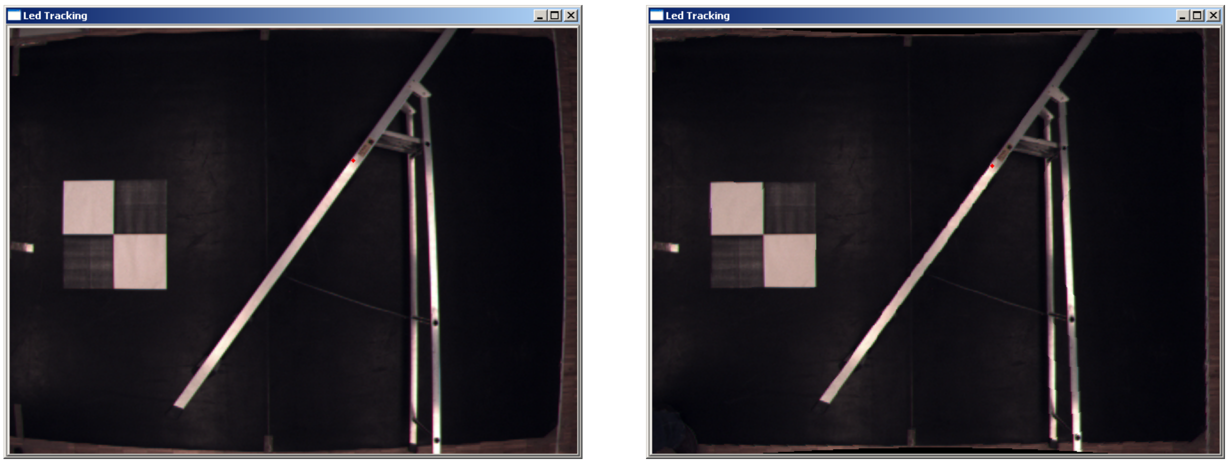


Figure 3.7: Example of image correction

Algorithm 4 Tracking algorithm

```
1: loop
2:     Grab a color image of 640x480 pixels from the camera
3:     Create a B/W image with the brightest red pixels in white
4:     $max\_pixel \Leftarrow 0$, $max\_x \Leftarrow 0$, $max\_y \Leftarrow 0$
5:     for $i = 1$ to $IMAGE\_HEIGHT - WINDOW\_HEIGHT$ do
6:         for $j = 1$ to $IMAGE\_WIDTH - WINDOW\_WIDTH$ do
7:             $pixel\_counter \Leftarrow 0$
8:             for $k = 1$ to $WINDOW\_HEIGHT$ do
9:                 for $l = 1$ to $WINDOW\_WIDTH$ do
10:                    $pixel\_counter \Leftarrow pixel\_counter + image[i * IMAGE\_HEIGHT + j + k * WINDOW\_HEIGHT + l]$
11:                end for
12:            end for
13:            if $pixel\_counter > max\_pixel$ then
14:                $max \Leftarrow pixel\_counter$, $max\_x \Leftarrow j$, $max\_y \Leftarrow i$
15:            end if
16:        end for
17:    end for
18:    Send $max\_x$ and $max\_y$ via TCP
19: end loop
```

3.4 CPG Implementation into the Microcontroller

The CPG model presented in Chapter 2 is implemented in a distributed fashion in YaMoR. Each module runs its own oscillator, whose set-point controls the position of the servo-motor. The modules also communicate together for the synchronization of the CPG. Furthermore, the remote PC, that runs the optimization algorithm, is able to fully control the functionalities of each module. All these features are coded in C language and compiled to ARM object code using WinARM (Thomas, 2004) environment, with the option `-O3` for the optimizations. The object code is then transferred serially into each LPC2138 microcontroller.

The firmware was developed upon a framework provided in the example folder of WinARM (`lpc2138_uart0_irq`). It encapsulates the low-level register programming and thus speeds up the software development. In the rest of the section, the relevant features of the firmware are presented.

3.4.1 Timer

In embedded applications, a time basis is often required to have an accurate control of the system. This need is even more accentuated when a regular oscillator output has to be generated. For this reason, `TIMER0` is used to generate interrupts each millisecond. The interrupt handler is as follows:

```
void __attribute__ ((naked)) tc0_cmp(void) ;

void tc0_cmp(void)
{
  ISR_ENTRY();

  timeval++;                // increment the timer at each interrupt
  T0IR = TxIR_MR0_FLAG;     // Clear interrupt flag by writing 1 to Bit 0
  VICVectAddr = 0;          // Acknowledge Interrupt

  ISR_EXIT();
}
```

`timeval` (on 64 bits) contains then the number of milliseconds elapsed since system startup. It is important to note here the way the interrupt handler is declared. The keyword `naked` is used to inform GCC[7] that `tco_cmp` does not need prologue/epilogue[8] sequences, which are manually introduced by the macros `ISR_ENTRY` and `ISR_EXIT`. This method ensures the compatibility of the code with the Thumb mode[9], which is however not currently used in the firmware.

[7]Gnu Compiler Collection.

[8]A function prologue prepares the stack and registers used by the function. The epilogue reverses these operations.

[9]The Thumb mode (specific to ARM) is a way of coding the assembler instructions on 16 bits, instead of 32 bits. Although the instructions have less functionality, the code density is improved. It seems that GCC generates incorrect prologue/epilogue sequences in Thumb mode.

3.4.2 UART

The microcontroller communication with the outside world passes through the `UART1` interface, whose `RXD1` and `TXD1` pins are connected to the corresponding serial pins on the ZV4002 chip of the Bluetooth board. Therefore, the microcontroller is able to send/receive SNP packets to/from other modules in the Bluetooth network.

The framework `lpc2138_uart0_irq` provides an interrupt driven UART management, both for the sending and the receiving. Nonetheless, the system used to become unstable non deterministically, especially in the sending part. For this reason, the sending is finally done without interrupts. The UART speed is set to the maximum 115200 baud.

UART Sending

For sending data through the UART (without interrupt system), a busy loop simply waits that the transmit register (`U1THR`) is empty (scrutation on `U1LSR`), before filling it with a new byte. In this case, care has to be taken that the timings are coherent, i.e. the time window is respected.

UART Receiving

The UART has a 16 bytes receive FIFO, programmed to generate an interrupt when it contains 14 bytes. When the interrupt is generated, an handler transfers the content of the FIFO into a software queue of 512 bytes, which is then read by the main code. In order not to miss incoming data, it is compulsory to use interrupts for the UART receiving. To avoid buffer overflow, timings must also be correctly ensured in this case.

3.4.3 Structure of the Main Control Loop

Instead of using an Operating System, the approach of a main control loop has been chosen. Thanks to the `timeval` variable, the loop is divided into two tasks `CPG_thread` and `rec_thread`, lasting exactly 5 milliseconds each. A strict control is achieved to avoid that a task overrun its time window (error message sent in this case).

`CPG_thread`

This task firstly updates the state variables of the oscillator using Euler integration[10] with a timestep of 10 milliseconds (`CPG_thread` repeats every 10 ms) and generates a new set-point. Θ (see Chapter 2), which increases linearly, is lowered by the biggest multiple of π below 1000 when it reaches this value.

The set-point is then saturated with respect to a user-specified limit (via a command from the PC), corresponding to angles that the servo-motor should not reach. The final angle is converted into a duty of a Pulse-Width Modulator (`PWM2`), which controls the position of the servo. `PWM2` has a cycle duration of 5 milliseconds.

The rest of the task is dedicated to the sending of data through the UART. In order to avoid an overrun of the time window (5 ms) and to lower the pressure on the Bluetooth board, data is sent using a round-robin cycle of 240 milliseconds, organized as in Figure 3.8.

[10]The most basic kind of numeric integration, based on $f(t+\Delta t) = f(t) + (\Delta t * f'(t))$.

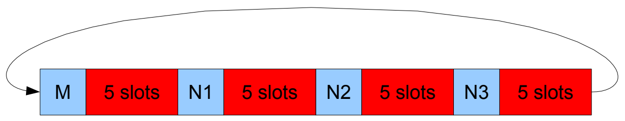


Figure 3.8: Organization of the time slots for sending through UART. A time slot starts every 10 millisecond, but only the first half can be used, the other is for the second task. In the slot `M`, the position of the set-point is sent to the master node (remote PC). In the slots `N1`, `N2`, and `N2`, r and Θ are sent to neigbor 1, 2, and 3 respectively

In addition to the sending cycle of Figure 3.8, short error messages (5 bytes) can be sent to the host if either one of the following occurs: a coupling lost, an UART error, or a time window overrun. Moreover, an alive packet (5 bytes) can be requested by the host to discover if the device responds.

In the worst case scenario, a maximum of 21 (coupling) + 5 (alive packet) + $3*5$ (errors) = 41 bytes are then sent in a window of 5 milliseconds. Using the UART at 115200 baud allows to send approximately 57 bytes during 5 milliseconds. Therefore, an overrun should not appear.

`rec_thread`

In this second task, the incoming bytes from the UART are processed during 5 milliseconds. The standard SNP data packet format is enhanced with an additional field that specifies one out of six different types of messages:

1. A message for entering the addresses of the neighbors (up to 3, for building the CPG structure).
2. A message for setting the 9 control parameters of the oscillator.
3. A message for setting the limits of the servo-motor (in radian).
4. A control message for defining the state of the motor, UART, oscillator, for allowing the module to send its set-point to the master, and for allowing the module to send error messages to the master.
5. A coupling message from one of the three possible neighbors (r and Θ received).
6. An alive message for allowing the master to know if the device responds.

The exchange of state variables between modules takes place every 250 milliseconds, which is larger than the 10 milliseconds of the integration step. Although r and Θ are kept constant during the inter-communication time, Auke Ijspeert discovered that the oscillations are not much perturbed, as long as the communication step is below 30 times the integration step (i.e. below 300 milliseconds). Nevertheless, when a module do not receive the state variables from a neighbor for more than 380 milliseconds (communication delays in the network or device failure), the corresponding coupling parameters are set to 0, for limiting the degradation of the oscillator output.

3.5 Implementation of the Optimization and Control Algorithms

In this section, the main software (`YaMoRHost`) running on a remote PC is detailed. It serves as a control of the modules and runs the optimization algorithms. The application is composed of several threads running in parallel, namely a textual user interface, a thread listening on TCP (for receiving the position of the LED), a thread listening on the serial line (for receiving feedback from the modules), a thread for the optimization algorithms, and a thread for displaying a feedback window. `YaMoRHost` communicates with the other modules via the UART of a spare Bluetooth board, connected to the serial port of the PC. All the settings concerning the modules are made via files. The different commands and configuration file structures are exposed in the directory of the application.

3.5.1 Control of the Modules

The first compulsory step for controlling the modules is to establish a Bluetooth Scatternet structure. `YaMoRHost` reads this structure from a *connection* file and sends the appropriate commands through the serial line of the PC to the Bluetooth transceiver. This latter is then responsible to build the Bluetooth network and will be known as the master node of the other modules. All the commands destinated to the YaMoR will be launched from this node and it will also receive the feedback messages. As discussed in Section 3.2, a delay between two connection commands is necessary, in order to let the Bluetooth pairing process operate. In the current version of `YaMoRHost`, the connection delay is set to the lowest working value, i.e. 7 seconds. The connection procedure could be improved if the alive message feature of the microcontroller was used. Indeed, the host application could wait on a reply from the microcontroller before processing the next connection command. The most stable connection structure that was experimented is a chain (see Section 3.2). In the other configurations, the Bluetooth chip used to reset itself because of overflow.

The other commands for controlling the modules correspond to what was presented in Section 3.4. When a serie of commands have to be sent from the master node to different devices, a delay of 1 second is inserted. Morevover, as the broadcast feature of the Scatternet protocol seems to be unreliable, the commands are sent iteratively.

In the Scatternet protocol, the Bluetooth devices send state and error messages by default. This functionality is suppressed after the initial connection, in order not to overload the Bluetooth network.

3.5.2 Fitness Function

An optimization algorithm repeatedly evaluates the performance of the robot for a given set of CPG parameters. `YaMoRHost` first sends the parameters to all the modules (1 second per module). A convergence time of 7 seconds is then waited. In the following step, the position of the LED (periodically received via TCP from the application presented in Section 3.3) is recorded. After 8 seconds, the new position of the LED is again recorded and the average speed between the two measurements is computed as the covered distance over the elapsed time. The result is a value in pixel per seconds (1 pixel = 0.0041 m approximately).

As the optimization algorithms presented in Chapter 2 are made for function minimization, the final value returned to the optimizer is normalized between 0 and 1, and a higher speed means a lower value. Therefore, the fitness function is defined as follows:

$$f(\vec{x}) = \frac{1}{avg\_speed + 1} \tag{3.1}$$

The optimization algorithms may sometimes come up with values that are out of the allowed range ([0, 1]). In this case, the value 1.01 is returned.

3.5.3 Implementation of Brent's Method

Brent's method has been implemented following the guidelines proposed in the Numerical Recipes (NR) book (Press *et al.*, 1988). The method assumes an initial bracketing of the minimum of the function is available. The risk of falling into a local optimum of the function can be reduced by defining our own initial bracketing procedure.

All the parameters are normalized between 0 and 1. The lower boundary of the interval is set to -0.1 and the upper boundary to 1.1, so as to include the limits in the search. The initial minimum is then randomly fixed in the interval. Therefore, at each iteration, the whole search space is available.

The stopping criterion of Brent's method is that the bracketing triplet is relatively small. The inequation given in NR is as follows:

$$\left| x - \frac{a+b}{2} \right| \leq 2.0 * tol * |x| - \frac{b-a}{2} \tag{3.2}$$

This means that a typical ending for the algorithm is to have an interval that is $4.0 * tol * |x|$ wide with x the minimum in the middle of the interval $[a,\ b]$.

A difference of 0.02 between a and b is sufficient, because we work in $[0, 1]$ and this is a percentage of another range. For instance, if the original range is $[0,\ 2\pi]$, an increase of 0.02 corresponds to an increase of 0.12 approximately. If the final $x = 0.5$, tol should be 0.01. On the other hand, if $x = 0.1$, then tol should be 0.05. We feel that the stopping criterion is somehow not very good (dependence on x is annoying). However, as many experiments were done with it, it was kept for practical reasons.

For the experiments, a tolerance of 0.05 is finally chosen. This means that the maximum number of iterations to converge can be estimated. For instance, if the minimum is $x = 0.5$, the final size of the interval will be 0.1, and the algorithm is started with an interval of size 1.2. If the golden section is always chosen (worst case), 6 iterations are needed. On the other hand, if $x = 0.1$, 9 iterations are needed.

For the future, a more adapted criterion would be as follows:

$$b - a \leq tol \tag{3.3}$$

This criterion would be independent of the absolute values of a, b, or x and thus be probably more interesting.

3.5.4 Implementation of Powell's Method

Powell's method has also be written with the NR model. The stopping criterion of the algorithm is as follows:

$$2.0 * (f(\vec{x}_i) - f(\vec{x}_{i+1})) \leq tol * (|f(\vec{x}_i)| + |f(\vec{x}_{i+1})|) \tag{3.4}$$

where $f(\vec{x}_i)$ is the minimum of the function at iteration i.

Since the fitness measurement is noisy (see below), it could happen that $f(\vec{x}_{i+1}) > f(\vec{x}_i)$. Therefore, the criterion is slightly modified to:

$$2.0 * |f(\vec{x}_i) - f(\vec{x}_{i+1})| \leq tol * (|f(\vec{x}_{i+1})| + |f(\vec{x}_i)|) \tag{3.5}$$

As for Brent's method, the criterion is surprising, because it is dependent on the value of the function return. The values of our fitness function are in the range [0, 1]. Imagine that 0.9 is found at one iteration and 0.8 at the next iteration, and the algorithm is requested to stop. Given the above equation, *tol* should be set to 0.12 or more. If we are in the case of one iteration leading to 0.2 and the next to 0.1, *tol* should now be set to 0.66 or more.

In this master project, the tolerance has been set empirically to 0.02, as it seems to be the most adapted value for our setup. A more interesting criterion would go as follows:

$$|f(\vec{x}_i) - f(\vec{x}_{i+1})| \leq tol \tag{3.6}$$

3.5.5 Noise on the Fitness Measurement

As can be seen on Figure 3.9, there is an inherent noise on the fitness evaluation. Indeed, instead of moving linearly at a constant speed, the robot has an alternance of rest and rapid accelerations. Depending when the position of the LED is recorded, the distance moved can thus be different.

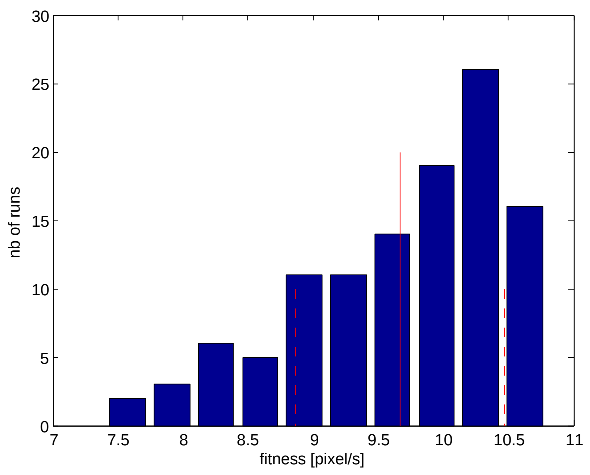


Figure 3.9: Repeated fitness evaluations for the same set of parameters. The continuous vertical red line is the mean, while the two dashed lines represent the standard deviation.

This noise has naturally an influence on the optimization algorithms. It can lead the algorithm in a wrong direction (local minimum), but can also extract it from a wrong trend. An heuristic, on top of Powell's method, is thus developed, such that the best gait that was discovered during the experiment is finally chosen.

Some tests on the mathematical function of Chapter 2 have been performed to show the influence of noise on Powell's method. A gaussian noise with a standard deviation ranging from 0 to 1 has been applied to the function. As can be seen on Figure 3.10, even with the highest noise factor, all the minima are located in the lowest region of the function. When a function with a local and a global optima is used, Figure 3.11 shows that a certain noise factor helps more points to go into the global optimum.

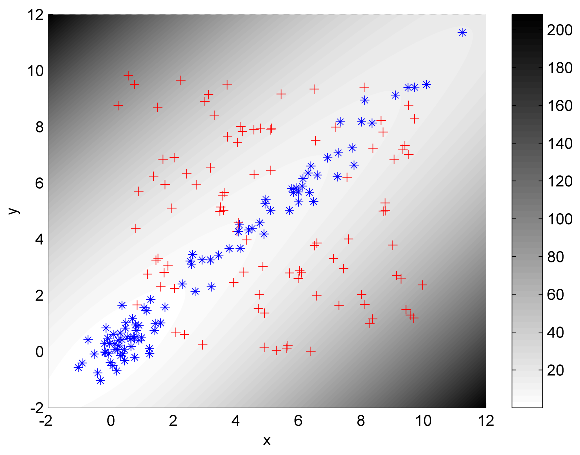


Figure 3.10: Example of noisy minimization. The function to minimize is $f(x, y) = z = \sqrt{x^2 + y^2} + (x - y)^2$. A gaussian noise with a standard deviation of 1 is applied to the function. The red crosses represent 100 starting points and the blue stars the final minima, which are all located in a good region.

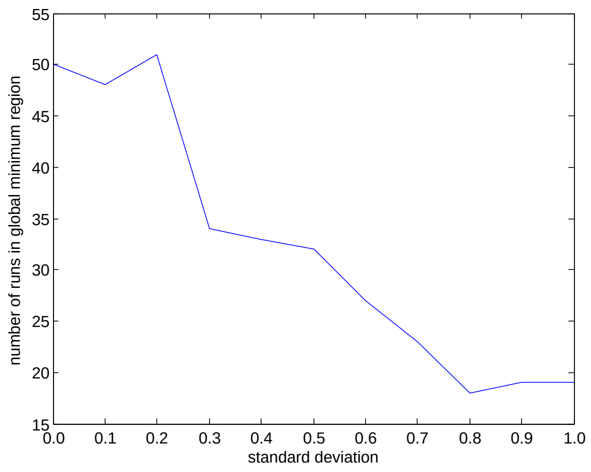


Figure 3.11: Example of noisy minimization on a function with 2 minima. The function to minimize is $f(x, y) = z = \sqrt{x^2 + y^2} + (x - y)^2 - x * \exp(-(x - 5)^2 - (y - 5)^2)$. 100 runs per standard deviation value are performed. A standard deviation in the range [0.1, 0.2] seems to bring more points in the correct final region.

3.5.6 Miscellaneous

Besides Powell's method and Particle Swarm Optimization, `YaMoRHost` can perform systematic search on a given set of parameters and do simple fitness evaluations. Furthermore, a useful graphical window can be displayed to follow the displacement of the LED on the left side, and the evolution of the fitness evaluations on the right side (Figure 3.12).

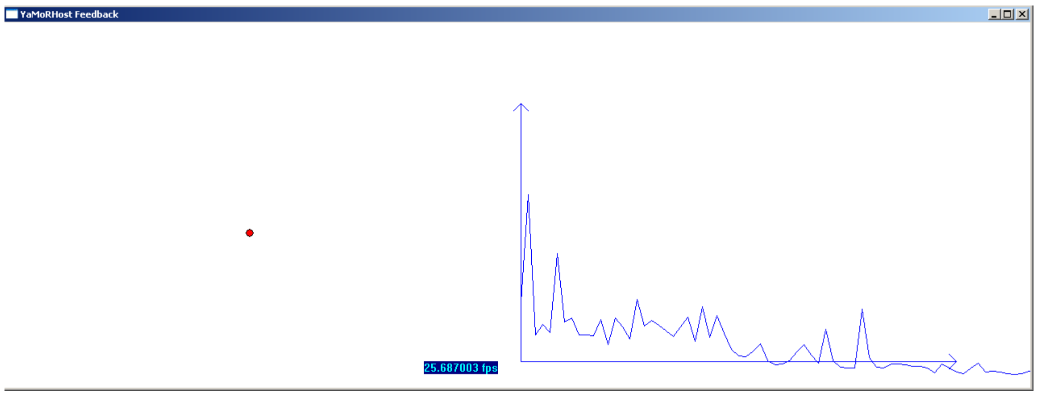


Figure 3.12: `YaMoRHost` feedback window.

`YaMoRHost` has been developed under Microsoft Visual C++ 6.0 with the Win32 Application Programming Interface (API). When taking the Visual project as a basis for future development, care should be taken that the compiler is using the multithreaded C runtime library. Furthermore, the precompiled header option should be avoided. For unknown reasons, Visual C++ sometimes automatically changes these settings when transferring the project onto another machine.

3.6 Implementation of the Simulation

It is sometime worth to have a simulation environment at hand for doing preliminary experiments or systematic tests. Therefore, the previously described experimental setup has been implemented in the Webots 5.1.11 robot simulator (Figure 3.13), using a framework from Yvan Bourquin[11] (`yamor.wbt`).

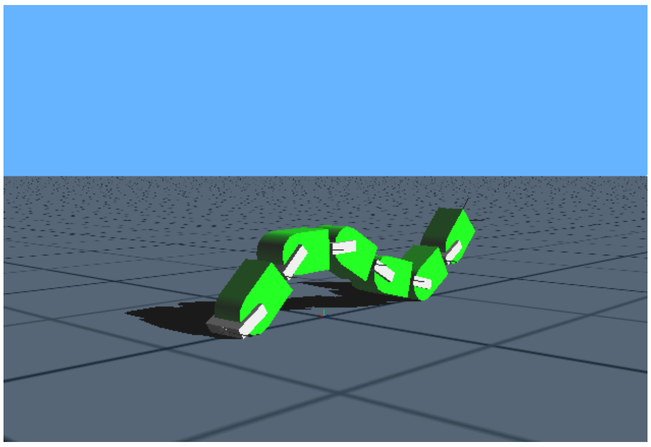


Figure 3.13: YaMoR in the Webots simulator.

Each simulated module runs nearly the same code as the microcontroller, except for the communication, which is done with the broadcast emitter/receiver system (Cyberbotics, 2007) from Webots. A supervisor node launches `YaMoRHost`, which is enhanced with the emitter/receiver system and another fitness measurement method. Instead of following the position of a LED, the position of a module in the middle of the robot is tracked (it can be easily extracted).

Although intensive work has not yet been accomplished in Webots, it could bring a lot of advantages in the future. For instance, when a new robot structure is designed, simulations could be run to test if the real experiments are worth. An optimization algorithm could also be first run in simulation and a slight adaptation could be done on the real robot. Finally, continuous learning, even in presence of a damaged module, should be facilitated.

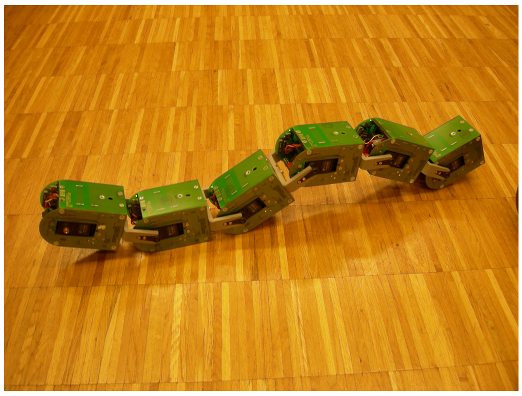
3.7 Robot Configurations

Systematic experiments were carried out on three different robot shapes: a snake, a tripod, and a quadruped. The robot is placed into a square of 2m by 2m with a rubber floor. A camera, fixed on the ceiling, covers the entire arena and tracks the position of a LED attached to one of the module (see Section 3.3). The optimization algorithms, running on another PC, use the position of the LED for estimating the velocity of the robot and for updating the parameters under optimization. These latter are then sent to the modules. In case the robot goes out of the field of vision of the camera, it is manually placed in the center of the arena. In this section, the CPG architecture and the mechanical structure of each robot is presented.

[11]Researcher at the BIRG.

3.7.1 Snake

The snake robot is made of five actuated modules and a passive one in the front, as can be seen on Figure 3.14. The CPG network is a chain of 5 oscillators that follows the mechanical structure of the robot. Each module is coupled with the neighbors it is mechanically attached to (bidirectional couplings). As this robot shape presents several symmetries, the number of parameters to optimize is dramatically reduced. For the proof of concept of our method, only 2 parameters are optimized, namely the shared amplitude $R = R_i$ and phase lag $\varphi = \varphi_{ij}$. R is restricted in the range $[0, \frac{\pi}{5}]$ and φ in $[0, \pi]$. All the offsets X_i are set to 0, ω_i to $\frac{3\pi}{5}$, and the coupling weights ω_{ij} to 1.



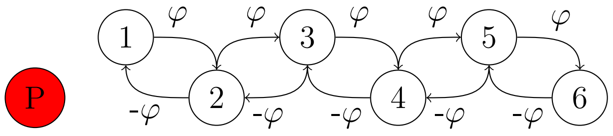


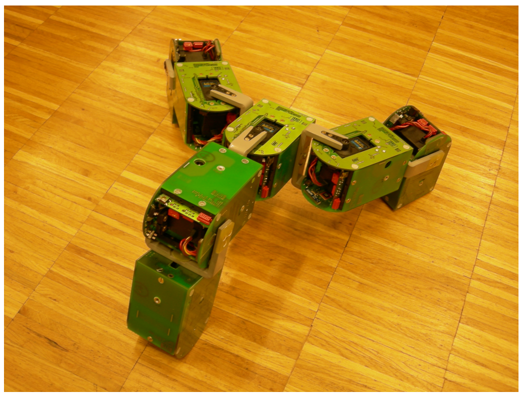
Figure 3.14: Mechanical structure of the snake robot (left), CPG architecture (right). The red P represents the passive module.

3.7.2 Tripod

The tripod robot consists of six active modules and a passive one in the front, as shown on Figure 3.15. Each limb has 2 oscillators coupled together. The inner one is also coupled with its 2 neighbors, following the mechanical structure. The symmetries are also widely exploited in this robot, since only 6 parameters are optimized:

- A shared amplitude $R1$ by all the outer modules (1, 4, 6), restricted in the range $[0, \frac{\pi}{2}]$.
- A shared offset X by all the outer modules, restricted in the range $[0, \frac{\pi}{2}]$.
- A shared amplitude $R2$ by all the inner modules (2, 3, 5), restricted in the range $[0, \frac{\pi}{4}]$ (to avoid collisions).
- A shared phase lag φ_{AB} between the 2 modules of each limb, restricted in the range $[0, 2\pi]$.
- A phase lag φ_1 between the inner modules 2 and 3, restricted in the range $[0, 2\pi]$.
- A phase lag φ_2 between the inner modules 3 and 5, restricted in the range $[0, 2\pi]$.

The phase lag φ_3 between the inner modules 5 and 2 is deduced from φ_1 and φ_2, such that $\varphi_1 + \varphi_2 + \varphi_3$ is a multiple of 2π. The offsets X_i of the inner modules 1 and 2 are set to $\frac{\pi}{3}$. All the frequencies ω_i are set to $\frac{3\pi}{5}$ and all the coupling weights ω_{ij} to 1.



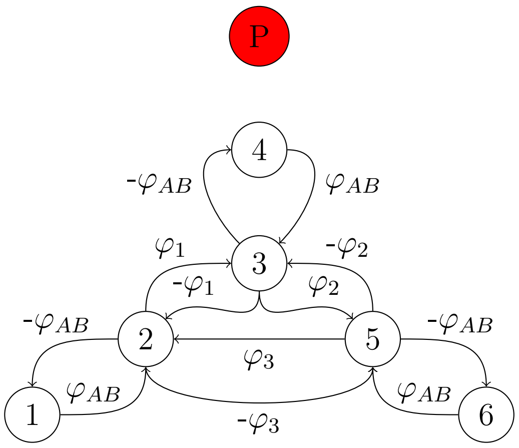
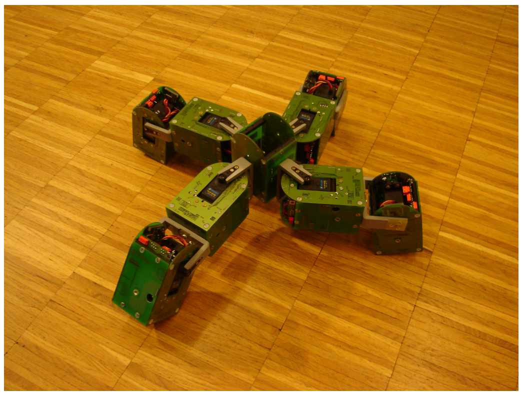


Figure 3.15: Mechanical structure of the tripod robot (left), CPG architecture (right). The red P represents the passive module.

3.7.3 Quadruped

The quadruped robot has eight active modules and a passive one in the middle, as depicted on Figure 3.16. The CPG architecture is similar to the one of the tripod, with one additional limb. This means that a supplementary phase lag φ_3 parameter is optimized (thus 7 dimensions), again restricted in the range $[0, 2\pi]$. All the offsets of the inner modules are now set to 0. The frequencies and coupling weights are as in the tripod. The phase lag φ_4 between 7 and 2 is such that $\varphi_1 + \varphi_2 + \varphi_3 + \varphi_4$ is a muliple of 2π.



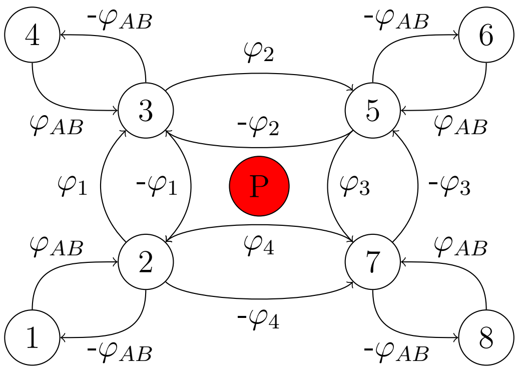


Figure 3.16: Mechanical structure of the quadruped robot (left), CPG architecture (right). The red P represents the passive module.

Chapter 4

Experimental Results

About half of the time dedicated to this master project has been spent on robot experiments. This compulsory step for validating the setup of Chapter 3 has required long hours of debugging to be able to present the results of this chapter.

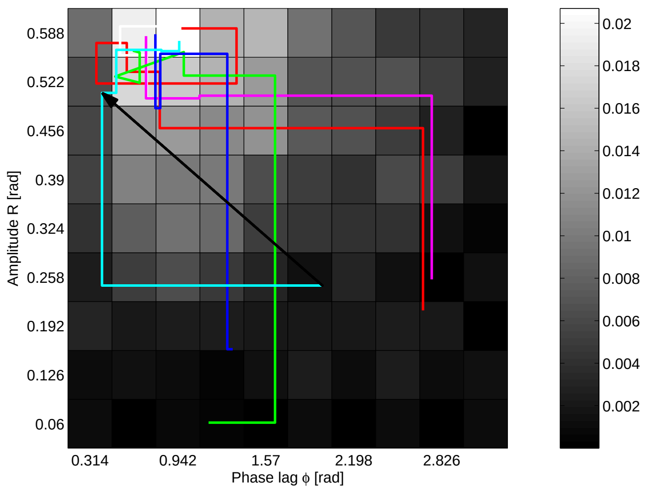


Figure 4.1: Systematic search on the snake robot. 81 fitness measurements have been carried out (9 equally spaced per parameter). The color bar on the right represents the speed of the robot in m/s. Some of the Powell experiments of Figure 4.2 are superposed above the area (colored lines). The highest velocity (about 0.02 [m/s]) appears when the amplitude $R = 0.588$ [rad] and the phase lag $\phi = 0.942$ [rad].

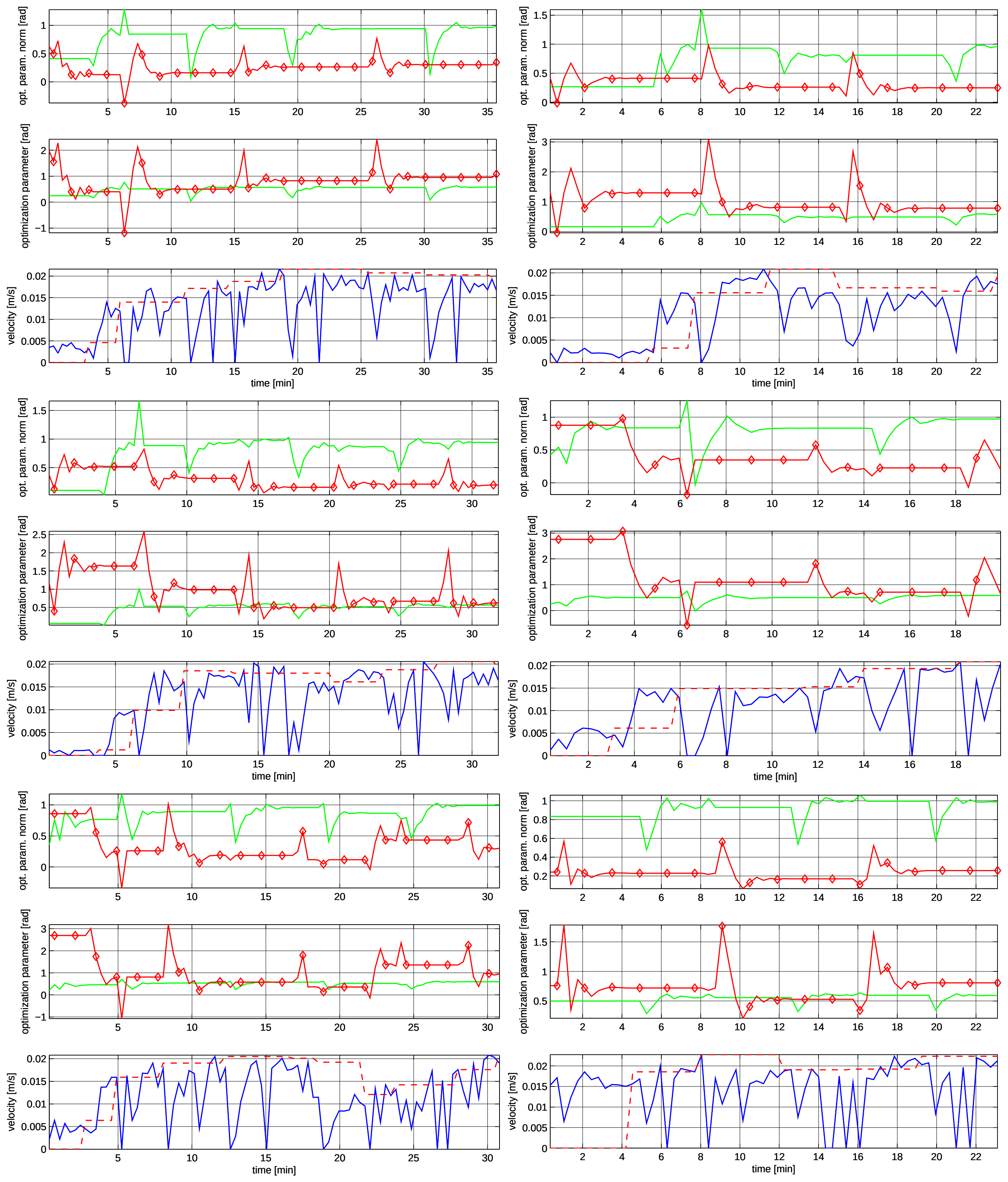


Figure 4.2: Powell's method on the snake robot. 6 different experiments are depicted here. The green line represents the amplitude R and the red line (with the squares) the phase lag ϕ. The parameters are first shown in their normalized form, between 0 and 1, and then in their real form. Below the parameters, the velocity of the robot in m/s is printed. The blue line represents all the fitness evaluations, while the red dashed line only the results of the line minimizations. The repeated two line minimizations, followed by the point extrapolation, are clearly visible.

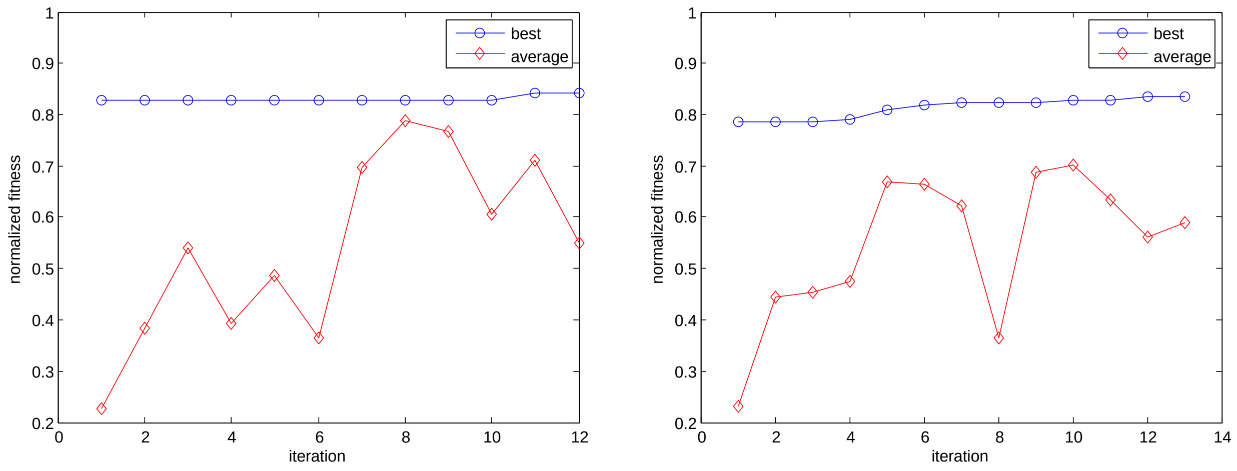


Figure 4.3: Particle Swarm Optimization on the snake robot. 2 experiments have been carried out. The blue line (circles) is the evolution of the best individual and the red line (diamonds) of the average of the population, made of 10 particles randomly spreaded in the search space. The weights pw and nw are set to 2.0. The best individual on the left ends up with a velocity of 0.022 m/s, an amplitude $R = 0.59$, and a phase lag $\varphi = 0.84$ (radians). The best individual on the right ends up with a velocity of 0.021 m/s, an amplitude $R = 0.59$, and a phase lag $\varphi = 0.63$.

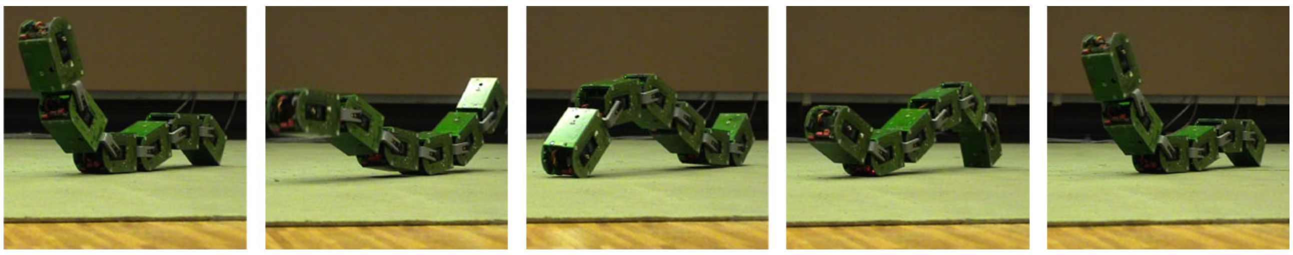


Figure 4.4: Snapshot of the evolved gait for the snake robot.

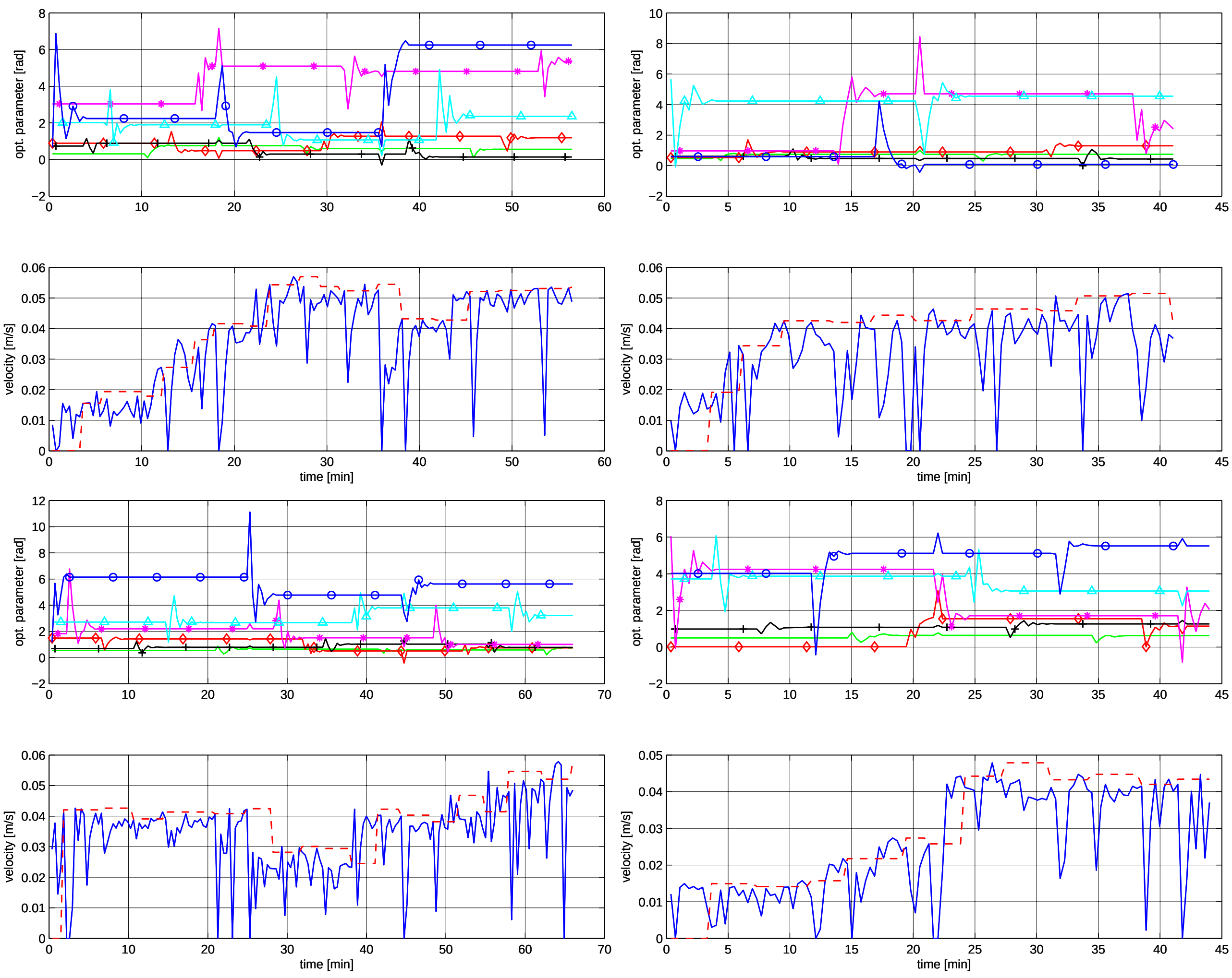


Figure 4.5: Powell's method on the tripod robot. The 4 most representative experiments are shown here. The parameters are presented with their real value in radians. The green line is R_2, the red line (diamonds) is R_1, the black line (crosses) is X, the purple line (stars) is φ_{AB}, the cyan line (triangles) is φ_1, and the blue line (circles) is φ_2. The velocity graph below has the same meaning as in Figure 4.2.

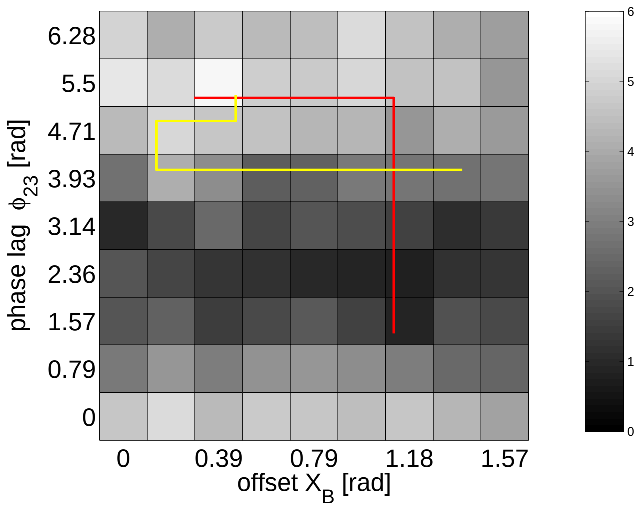


Figure 4.6: Systematic search on the tripod robot. The parameters X and φ_2 are explored on a 9x9 search space. The other parameters are fixed to some good values. The paths superposed above the area represents two Powell minimizations. The velocity is in pixel/s, where 1 pixel is about 0.0041m. The highest velocity is about 0.02 m/s, when $X = 0.39$ [rad] and $\varphi_2 = 5.5$ [rad].

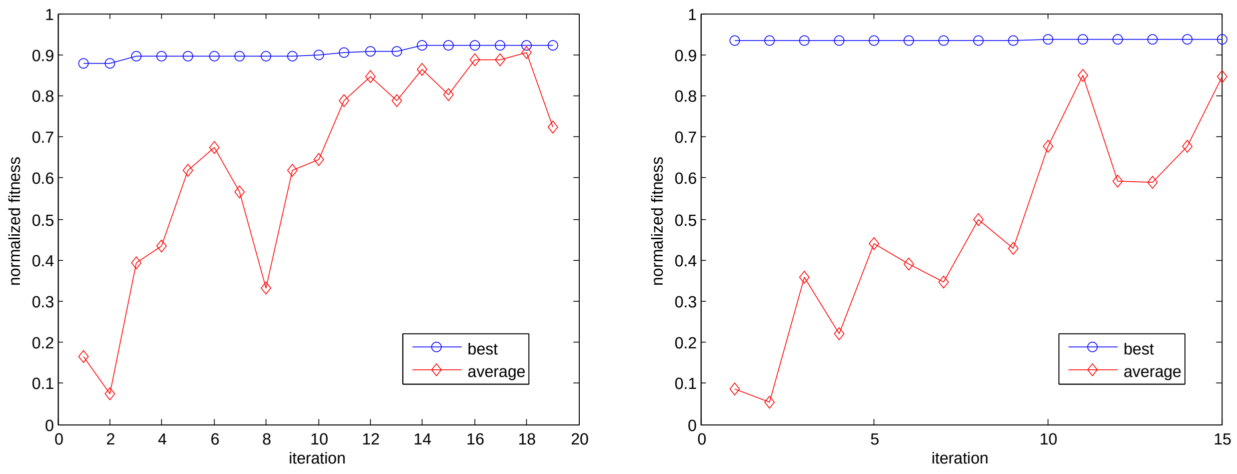


Figure 4.7: Particle Swarm Optimization on the tripod robot. 2 experiments have been carried out. The blue line (circles) is the evolution of the best individual and the red line (diamonds) of the average of the population, made of 10 particles randomly spreaded in the search space. The weights pw and nw are set to 2.0. The best individual on the left ends up with a velocity of 0.049 m/s, $R_1 = 0.99$, $R_2 = 0.62$, $X = 0.39$, $\varphi_1 = 4.9$, $\varphi_2 = 4.74$, $\varphi_{AB} = 2.55$ (radians). The best individual on the right ends up with a velocity of 0.06 m/s, $R_1 = 1.05$, $R_2 = 0.7$, $X = 0.64$, $\varphi_1 = 2.46$, $\varphi_2 = 0.13$, $\varphi_{AB} = 5.82$.

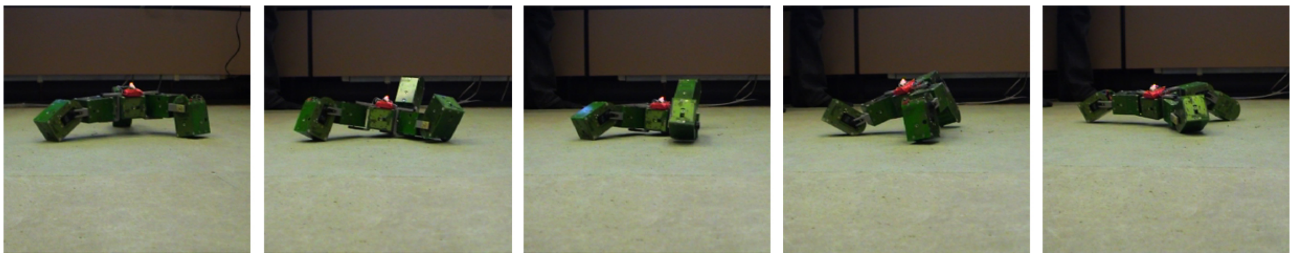


Figure 4.8: Snapshot of the evolved gait for the tripod robot.

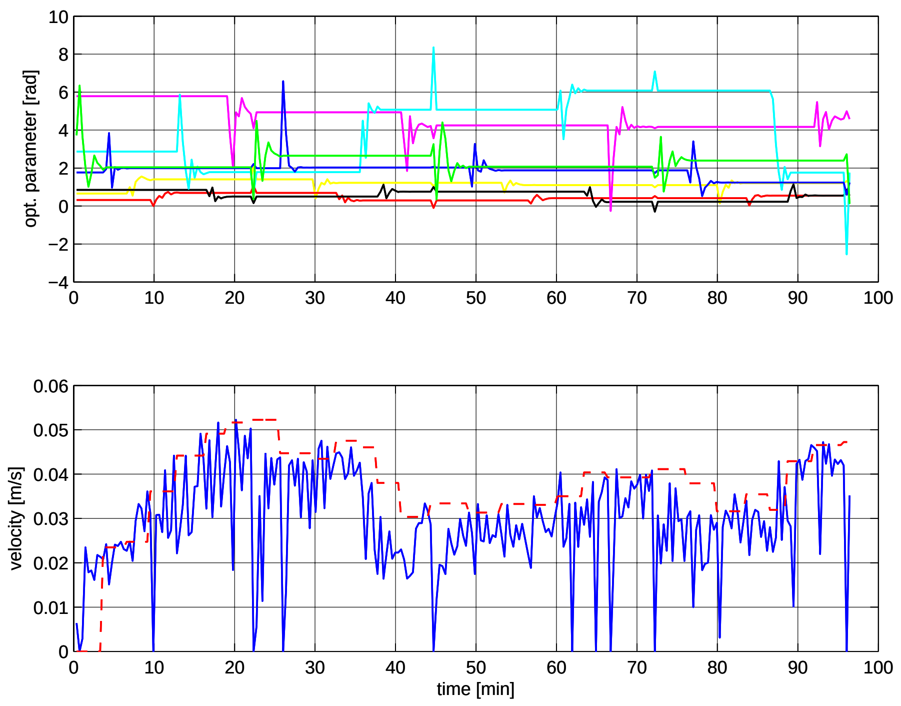


Figure 4.9: Powell's method on the quadruped robot. The meaning of the colored lines in the upper graph is the same as in Figure 4.5, added with a yellow line for φ_3. The velocity graph has the same meaning as before.

Chapter 5

Discussion

In this chapter, the results from Chapter 4 are commmented and discussed. Each robot shape is analyzed separately and the validity of our approach in general is questioned.

5.1 Snake Robot

The snake robot is a perfect example for proving the suitability of our Powell's method implementation. Indeed, all the experiments ended in the same minimum region, that was proven to be minimum by the systematic search (Figure 4.1). The noise (see Chapter 3) implies that the final parameters values are slightly different. This appears especially on the phase lag φ.

Since the fitness function has only one global optimum and 2 parameters are optimized, Powell performs nicely in these experiments. After 10 minutes (one iteration of Powell), it has nearly found the final minimum. At each new line minimization, it can be seen that most of the range is covered and not only a small interval around the previous minimal value.

The two experiments of PSO have found approximately the same values as Powell, i.e. a velocity of 0.02 m/s, an amplitude $R = 0.59$ and a phase lag φ in [0.6, 0.9]. Nevertheless, in term of speed of convergence, Powell outperforms PSO on this problem, as can be seen on Figure 5.1.

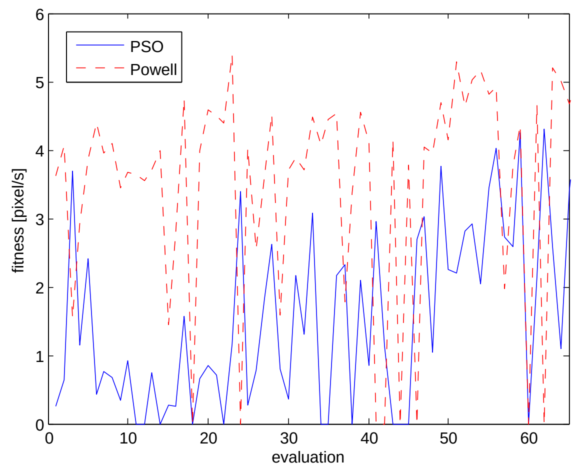


Figure 5.1: Comparison between Powell and PSO in term of speed of convergence in the snake robot. After 25 robot evaluations, Powell has already found the final minimum, while PSO is much slower.

5.2 Tripod Robot

With the tripod robot, encouraging results have also resulted from the Powell's method. The fastest gaits appears generally after 25 minutes. Instead of going forward, the robot has a tendency to move into a circle. This is probably due to the its structure.

As suggested by Figure 4.6, there seems to be several optima for the fitness function. This may explain why the final parameters are sometimes far from each other. In Figure 4.6, the two Powell experiments ended in the same correct minimum region.

PSO also found comparable results in term of velocity, i.e. around 0.06 m/s. On one experiment, it even jumped to 0.09 m/s, but this is probably the noise. Powell is again faster on this problem, as shown in Figure 5.2.

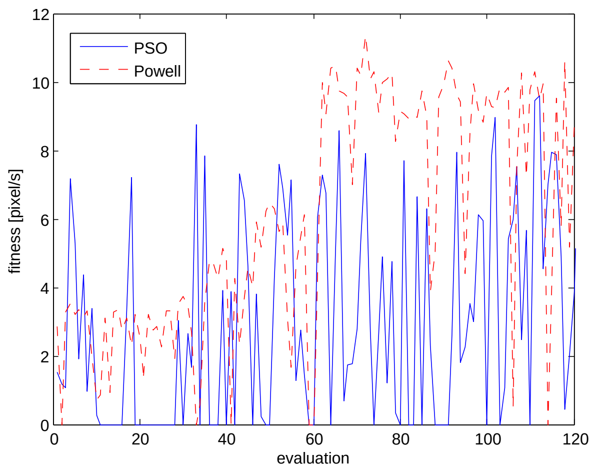


Figure 5.2: Comparison between Powell and PSO in term of speed of convergence in the tripod robot. After 60 evaluations, Powell has nearly the final minimum, while PSO is again below.

5.3 Quadruped Robot

The quadruped robot could be fully tested only at the end of the project. Therefore, only one significant experiment is shown (Figure 4.9). In this latter, Powell seems to work nicely. Indeed, although 7 parameters are optimized, a good gait is discovered after 20 minutes.

In this example, the effect of noise is clearly visible. After the good gait with a velocity of 0.05 m/s has been found, there is a whole suboptimal period before reaching a higher value. This is due to a changing of one of the phase lag into a region near a local minimum. For this reason, it seems important to select the best minimum seen so far when the algorithm is stopped.

In the quadruped robot, as in the tripod, the motion is most of the time circular. This could be probably avoided by changing the CPG structure.

Chapter 6

Conclusion and Future Work

This master project has shown the suitability of the Powell's method for doing online learning of locomotion on modular robots based on a Central Pattern Generator (CPG) architecture. A distributed CPG has first been implemented in the modular robot YaMoR. A complete control system for performing the optimization has then been built. Powell's method has been successfully tested on three different robot structures. It has then been compared with Particule Swarm Optimization (PSO), a stochastic algorithm. In all the experiments, Powell's method could reach a good gait faster than PSO. Finally, a simulation environment, that mimics our experimental setup, has been prepared and tested.

The efficiency of Powell's method remains to be proven in higher dimensional spaces. Some preliminary experiments have been done in simulations with the snake robot and open parameters (no symmetries). Nevertheless, no satisfactory gait emerged.

The online learning could be achieved in two steps. A quick global algorithm, such as PSO, could be used to determine the region containing a satisfying minimum. Powell's method could then be launched from a starting point in this region. We believe this would speed up the convergence time and avoid to fall into local minimum.

Life-long learning has not yet been experimented on YaMoR. This should be done quite easily on the basis of our work. Powell's method could be started whenever the velocity of the robot would fall below a certain limit. For instance, we could imagine that the quadruped robot would lose one of its outer modules and that the robot could adapt a new gait rapidly to this situation.

The modular robot could store the learned CPG parameters for a given structure in memory. Whenever the robot shapes would change, it could start Powell's method from this stored solution.

Finally, the control software `YaMoRHost`, that supervises the experiments, should be enhanced with a complete Graphical User Interface (GUI) in the future. At the current time, it is still arduous to work with it, because of its cryptic configuration files.

Bibliography

Arredondo, R. (2006). *Design and simulation of locomotion of self-organising modular robots for adaptive furniture.* Master's thesis, EPFL.

Bloch, G. (2006). Quand la table devient chaise. *Le Temps*, page 26.

Bray, J. and Sturman, C. (2000). *Bluetooth: Connect Without Cables.* Prentice Hall PTR.

Crow, B., Widjaja, I., Kim, L., and Sakai, P. (1997). Ieee 802.11 wireless local area networks. *Communications Magazine, IEEE*, **35**, 116–126.

Cyberbotics (2007). *Webots Reference Manual.*

Delcomyn, F. (1980). Neural basis of rhythmic behavior in animals. *Science*, **210**, 492 – 498.

Ijspeert, A. and Kodjabachian, J. (1999). Evolution and development of a central pattern generator for the swimming of a lamprey. *Artificial Life*, **5**, 247–269.

Kamimura, A., Kurokawa, H., Yoshida, E., Tomita, K., Kokaji, S., and Murata, S. (2004). Distributed adaptive locomotion by a modular robotic system, m-tran ii from local adaptation to global coodinated motion using cpg controllers. In *International Conference on Intelligent Robots and Systems.*

Kennedy, J. and Eberhart, R. (1995). Particle swarm optimization. In *Neural Networks.*

Kurokawa, H., Kamimura, A., Yoshida, E., Tomita, K., Kokaji, S., and Murata, S. (2003). M-tran ii: metamorphosis from a four-legged walker to a caterpillar. In *Intelligent Robots and Systems, 2003. (IROS 2003).*, volume 3, pages 2454–2459.

Marbach, D. and Ijspeert, A. (2006). Online optimization of modular robot locomotion. In *Conference on Mechatronics and Automation.*

Microchip (2003). *PIC16F87XA Data Sheet.*

Moeckel, R., Jaquier, C., Drapel, K., Dittrich, E., Upegui, A., and Ijspeert, A. (2006). Exploring adaptive locomotion with yamor, a novel autonomous modular robot with bluetooth interface. *Industrial Robot*, **33**, 285–290.

Philips (2005). *LPC213X User Manual.*

Press, W., Teukolski, S., Vetterling, W., and Flannery, B. (1988). *Numerical Recipes in C.* Cambridge University Press.

Segars, S., Clarke, K., and Goudge, L. (1995). Embedded control problems, thumb, and the arm7tdmi. *IEEE Micro*, **15**, 22–30.

Shik, M., Severin, F., and Orlovskii, G. (1966). Control of walking and running by means of electrical stimulation of the mid-brain. *Biofizyka*, **11**, 659–666.

Thomas, M. (2004). Winarm environment. http://www.siwawi.arubi.uni-kl.de/.

Xilinx (1999). *Spartan-3 FPGA Familiy: Complete Data Sheet.*

Xilinx (2004). *MicroBlaze Processor Reference Guide.*

Yim (1994). *Locomotion with a Unit Modular Reconfigurable Robot.* Ph.D. thesis, Stanford University Mechanical Engineering Dept.

Zeevo (2004). *ZV4002 Data Sheet.*

Zykov, V., Mytilinaios, E., Adams, B., and Lipson, H. (2005). Self-reproducing machines. *Nature.*