Imperial College of London

Research Project Report

Learning walking skills for Modular Robots

Author:
Clement Jambou

Supervisor:
Pr. Murray Shanahan

April 2014

IMPERIAL COLLEGE OF LONDON

Abstract

Department of Computing

Master of Advanced Computing

Learning Walking Skills and Laws of Command for Modular Robots

by Clement Jambou

In their design Modular Robots present many advantages when facing a task in a difficult environment. Among these advantages the way they could potentially adapt to this environment and reconfigure themselves when facing different tasks makes them the favorite candidate for many applications, such as space exploration. However controlling the swarm of robots from high-level commands remains a tricky problem, especially since this problem is non-linear and contains many degrees of freedom. Therefore it is resistant to classical approaches used in automation when dealing with Robotics motion control.

The goal of this project is to present a survey of different control models and learning algorithms that proved to be efficient on similar problems. To compare these methods, we introduce a benchmark simulation built on the ODE Simulator and publicly available on Github. In this project we also apply for the first time Liquid State Machines to this particular problem of modular Robot locomotion.

Contents

List of Figures

Chapter 1

Related Work

1.1 Overview

For over two decades, motion learning for complex structure has been a research focus. There has been many different approach to this problem, but it usually involves a simulated environment, where the creature can experiment with and get feedback from. The general problem is for this creature to learn by itself how to move in this environment in order to maximize a certain quantity such as its speed or distance with a certain amount of energy. The creature can have different ways of controlling its body using actuators to control the angle of its joints, (or even simulated muscles) and eventually some sensors to get feedback from the world. The goal being to generate the control function $\alpha_i(t, state)$ for each actuator's degree of freedom, that link time t and eventually the state of the creature to a command that can be send to the actuator (an angle for a servomotor, or command to a motor, an electrical signal to a simulated muscle ...). A widely used approach so far, which makes sense from a biological perspective is to implement smart actuators that can be linked directly to a sensor and modify the command with a low-level control. One of the examples of this behaviour is the muscle elasticity, which will alter the consequence of a specific command depending on the state of the muscle. Another example is the Proportional-Integral-Derivative controller (PID) of a servomotor. It is then possible to build a model that behaves as an open-loop from a high level perspective, but which actually shows robustness due to this low-level control loops. Under this assumption, we have the function $\alpha_i(t)$ that are only depending on parameter t. Finding the set of these functions remains an optimization problem in an infinite dimensional space. Therefore it is useful to make another assumption, which is that these functions are periodic. This make sense when we observe the movements of animals in the nature. In fact we can even consider the paradigm of oscillation based movements in the nature and apply it to the $\alpha_i(t)$ functions and write them using Fourier decomposition.

$$\alpha_i(t) = \sum_{k=1}^{N} b_{ki} * sin(kt) + \sum_{k=1}^{N} b_{-ki} * cos(kt) + b_{0i}$$

. With this decomposition, we transform our infinite dimensional research space (of functions) to a finite one containing the b_{ki} $(-N <= k <= N)$ coefficients. The parameter N is a restriction over the harmonics and can be seen as a precision parameter that determines how near we can get from any periodic function. This space of research remains big and does not reflects in its structure any of the physical interaction that can exist between two actuators (symmetry, graph structure) of the creature. Therefore, some of the following related works use different techniques to reduce the dimension of the space and also different learning techniques to obtain a solution.

1.2 Karl Sims Creatures

The first remarquable examples of modular robotics learning to evolve in a 3D simulated world is due to Karl Sims's Creature in 1994 [1]. In his work, the structure of the creature evolves at the same time as the control system. One way of reducing the search space of Fourier coefficients is to create a graph that generate oscillations (much like the brain of animals does). The neural network used in Karl's creature takes as input a set of values from different sensors and each node can perform a specific function such as sums, products, logical and trigonometric functions to generate the oscillations of the structure. The physical shape of the creature, is built from a genotype and informations to grow a creature from the genotype.

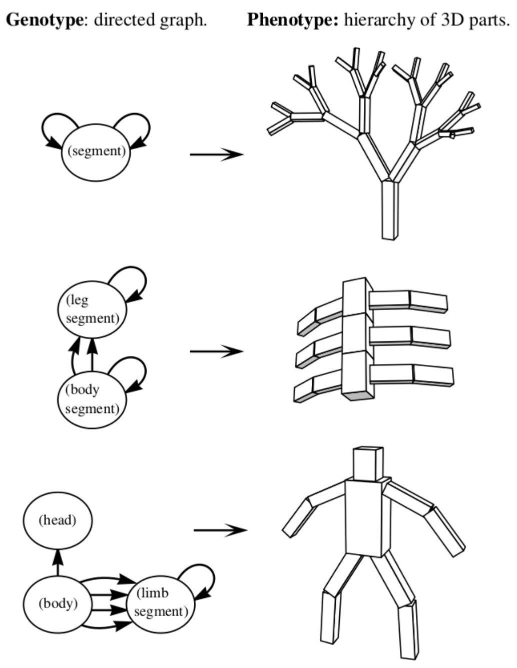
Genotype: directed graph.
Phenotype: hierarchy of 3D parts.
(segment)
(leg segment)
(body segment)
(head)
(body)
(limb segment)

FIGURE 1.1: Designed examples of genotype graphs and correspond- ing creature morphologies.

Karl Sims uses Genetic Algorithms in order to optimize all the parameters of the control graph and the structure. The mutation and cross-over steps of the Genetic Algorithm have been implemented for updating the population of genotypes. At each evaluation step, creatures are grown from a genotype and compared for a specific task (walking, jumping, swimming).

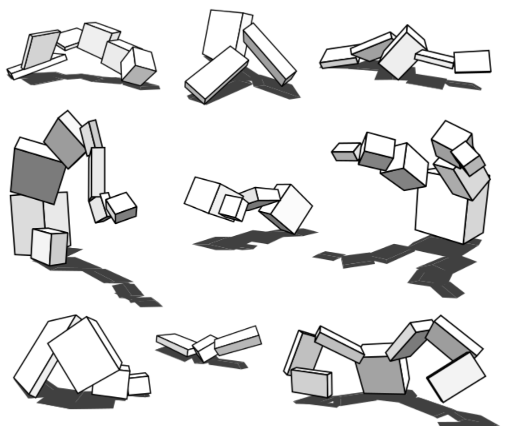

FIGURE 1.2: Karl Sims Creatures evolved for walking

1.3 Bio-Robotics lab at EPFL

In the last ten years, the bio-robotics lab of Ecole Polytechnique lauzanne (EPFL) has led breakthrough in modular Robotics, both in the Mechanical design and the control software. Central Pattern Generators are a widely use model to create oscillation for problems such as modular Robotics locomotion. In their work Sproewitz et al [3] focus on online learning parameters fron a Central Pattern Generator (CPG). A CPG is a graph that relates directly to the physical structure in order to generate oscillation (see description in Learning Models). They use a gradient-free downhill method: Powell's method, to learn the parameters of the CPG. In their design CPGs follow the same concept introduced by Karl Sims, where the physical shape is used to produce a similar graph for control purposes. However, for CPGs the neurons are only oscillators.

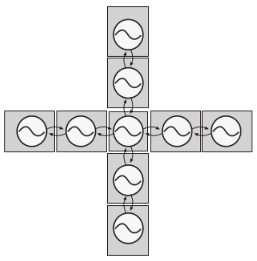

FIGURE 1.3: Central Pattern Generator Architecture

The Bio-Robotics lab also implemented physical blocs in order to build a swarm of modular Robots. The lab produced different prototypes to achieve this goal such as the YaMoR (Yet another Modular Robot) or Roombots module

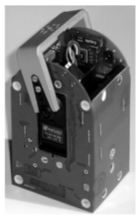

FIGURE 1.4: Modular Robots implementation

1.4 Flexible Muscle-Based Locomotion for bipedal creatures

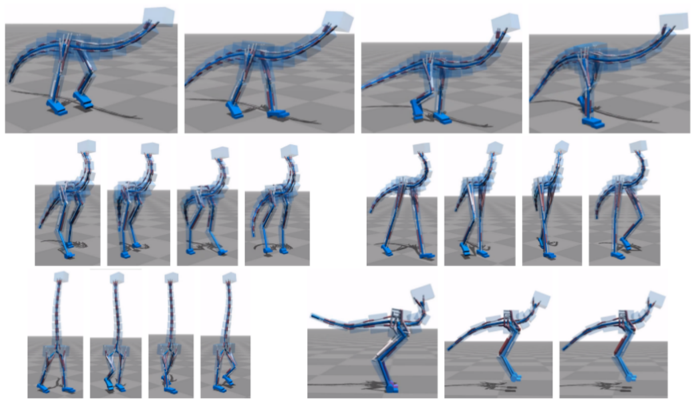

Figure 1.5: Synthesized walking for a bipedal creature

More recently, the progress of computational power and the growing interest of building humanoid robots for different tasks in non-friendly environment, such as the initiative from Virginia Tech to build a disaster response robots, or the growing demand of the animation movie and video games industry led to new results. Geijtenbeek et al [4] used muscle-based actuators and optimize at the same time the routing of the muscles and the control of the muscle activation from a musculoskeletal model. Though this approach is for graphic purposes (SIGGRAPH Conference) and it uses a musculoskeletal model, they follow the same method : a specific control model (simulation of the physical response and interaction of muscles) and a learning algorithm. In this paper, they use covariance matrix adaptation as a the optimization method [**?**]. This method updates a population of solutions by building new generations at each step based on a normal distribution, which mean and covariance are estimated from the evaluation of the previous generation. Their results showed robust locomotion for different speed, target directions and small ground variations in terrain.

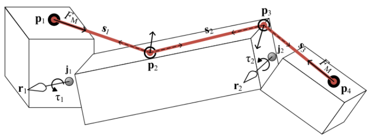

Figure 1.6: Example of Muscle Path

Chapter 2

Simulation Environment

The first task of my project was to build a simulator to be able to get feedback from a simulated world. The goal of the simulator is to provide an environment which is governed by physics laws. In this project such physics laws are gravity, friction and collisisons. Combining these phenomena on complex structures can lead to situations that are difficult to predict, especially since it has a chaotic behavior. Two movements of a structure in such a physical world, though they differ just a little, can lead to very different outcomes.

2.1 Physics Engine

In order to simulate the behavior of complex shapes and bodies in a simulated world, a physical engine is required. This kind of physical engine is used in a lot of different areas, for instance the film and video games industry, to simulate destructions or complex situations where placing all bodies by hand would be too difficult. I tested two different librairies that provide tools to simulate these complex situtations : `bullet` and `ODE` (`Open Dynamics Engine` [5]). The first part was to simulate the static part of the world, which is the ground. For now, I used a plane, with a friction coefficient, but we can imagine testing the creatures on different surfaces, (not necessarly plans)to test there reaction to a difficult environment. Also, in order to simplify the creature, so that the modules are simple shapes, all the robots are only composed of cubes (like the MIT project [6], linked by different type of joints. The idea behind this is to make the creature depending on a really simple implementation of these modules, that can benefit from a chain manufactoring. ODE and bullet also provide tools to simulate joints (adding constraint on the relative movements of different bodies) and also to animate them simulating motors. I choose ODE for its simplicity to control different joints using this motors. In order to simulate a servomotor properly, it is for instance possible to set a maximimum torque, boundaries within the degrees of freedom of the joint, and give a command to the motor.

All the calculation of the physics interaction is computed periodicly by setting the time to wait between two calculations. Setting this parameter can be tricky : if we set it value that is too small, then the simulation will slow and it will take time to compute a 10 seconds simulation of the world. On the contrairy, a large value for this parameter can lead to errors in the movement of the simulated bodies, especially if there are bodies with a high velocity in the simulation.

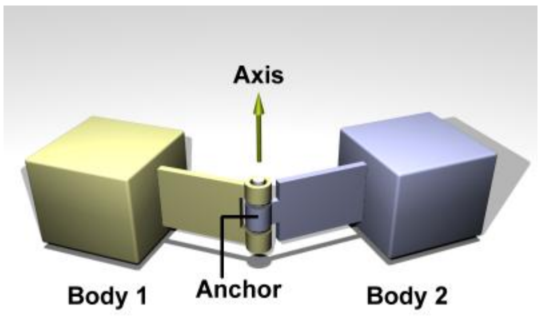

Figure 2.1: A 3D representation of a Hinge Constrain

2.2 Visual rendering

A good thing about the physical engine is that it is completly separated from the rendering part. That way, we can run all the simulation without watching the results which would recquire additional computation and slow down the learning process. But it is also necessary to see the result, especially for the purpose of debuging the simulator. Carnegie Mellon University developps a framework called panda3d ([7]), that integrate both a rendering librairy using OpenGl and different physics engines (ODE and bullet). All these tools are developped in C/C++ with binding for python, which makes it an easy tool to create animation movie, games or simulations. They provide very useful tutorial to get started using these tools on their website.

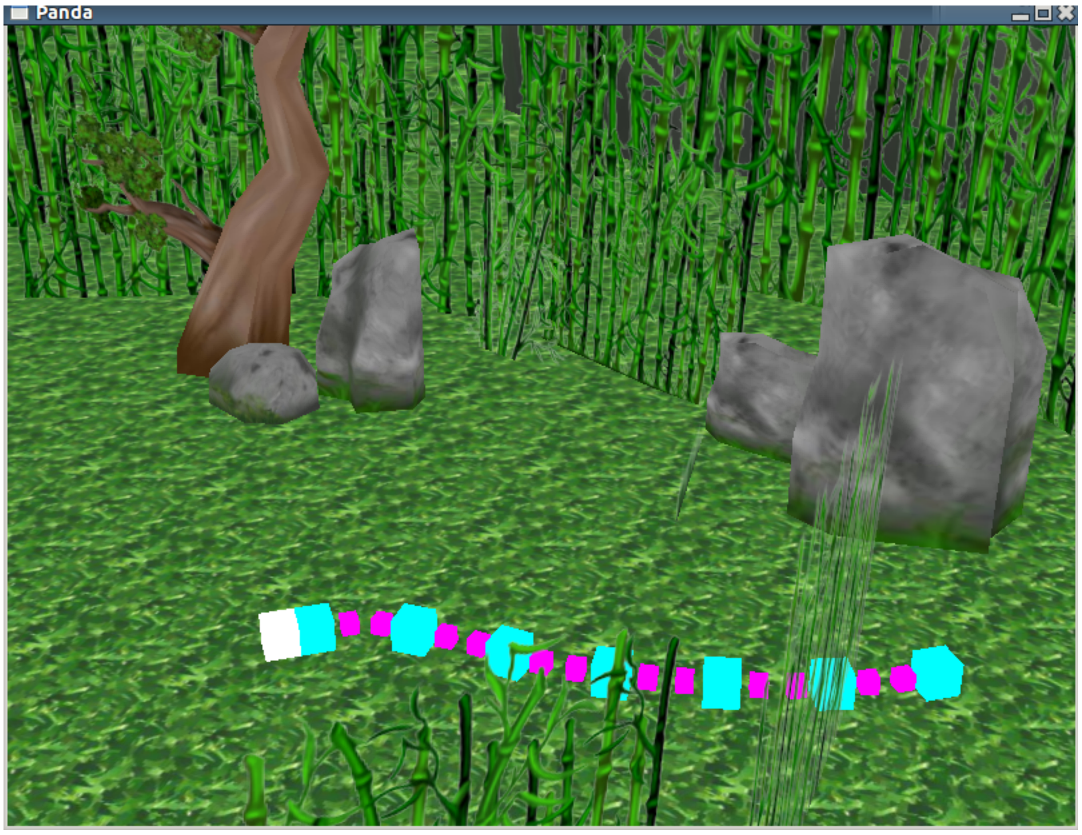

FIGURE 2.2: 3D Rendering of a snake in the environment

2.2.1 The coordinate system

In order to represent all the objects that we manipulate in the simulation (physics and rendering), panda3d, as most of 3D software, uses a system of global/local coordinate and a tree architecture. Each element of the tree is represented in the coordinates of the father. In order to keep a coherence with this architecture, it is natural to use a graph to represent the modular structure of the Robots in this project. Panda3d also uses 4-by-4 matrices to represent the transformation of a node to its one of its children. These transformation are classical in 3D representation. They combine the benefits of 3 by 3 matrices that represent functions ($\mathbb{R}^3 \to \mathbb{R}^3$) that can be interpreted as the set of combined rotations and homotetia, where the matrix multiplication is the composition of such transformations. But manipulating translations requires to use 4 by 4 matrices(multiplying any matrix by the vector $(0, 0, 0)^t$ will not change this vector) instead of 3 by 3 thatway the properties of the product are kept, which makes it a very useful tool to manipulate 3d objects. These calculation can run on a GPU as this is the kind of calculation GPU are designed for. For instance, if the children of the node are translated of a vector $(1, 0, 0)$, then we can use a function on the object representing the Matrice (TransformState in panda3d) to change the matrice and add the translation. The same goes for giving a certain orientation to the object using quaternions to represent the 3D orientation and add this to the 4 by 4 matrix.

$$T = \begin{pmatrix} \cos\alpha\cos\beta & \cos\alpha\sin\beta\sin\gamma - \sin\alpha\cos\gamma & \cos\alpha\sin\beta\cos\gamma + \sin\alpha\sin\gamma & x_t \\ \sin\alpha\cos\beta & \sin\alpha\sin\beta\sin\gamma + \cos\alpha\cos\gamma & \sin\alpha\sin\beta\cos\gamma - \cos\alpha\sin\gamma & y_t \\ -\sin\beta & \cos\beta\sin\gamma & \cos\beta\cos\gamma & z_t \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

FIGURE 2.3: Transformation Matrix for 3D bodies

2.3 Structures of the Robots as a Graph

We can represent a Robot as a graph, where each node is a component of the robot. In this project there is four types of components :

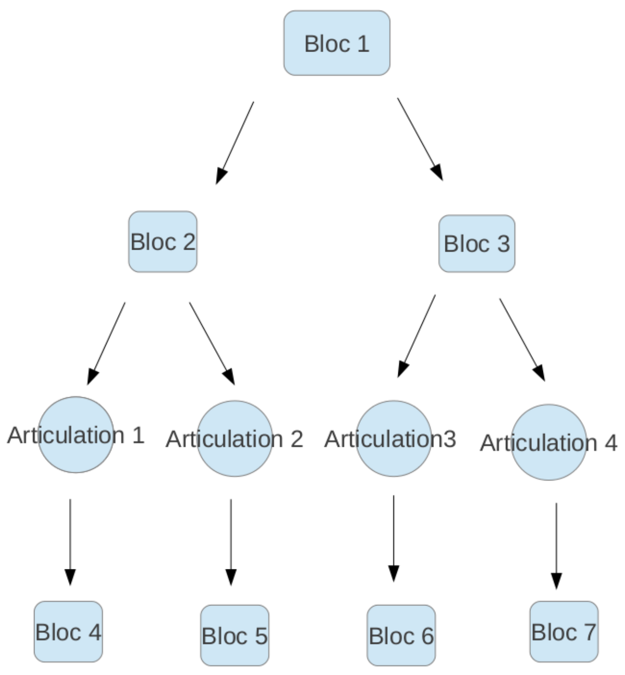

FIGURE 2.4: Structure described as a graph

- the head: which has a cube shapes and 6 sons (one for each faces), There is only one head per structure.
- a structural block: This is also a cube, also with 6 sons (or edges in the graph) for all faces.
- a hinge joint: The hinge is composed of two small cubes, separated by the joint whith one degree of freedom.
- a vertebra : a vertebra is very much like a hinge but has two degrees of freedom (two angle directions) with smaller range, but bigger maximal torque.

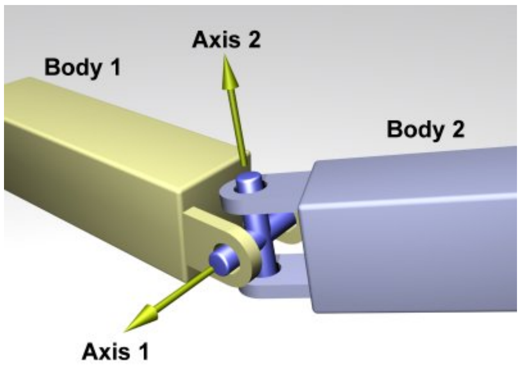

Figure 2.5: A 3D representation of a Vertebra Constrain

This Structure is represented in python with an object called MetaStructure, with very simple function to move within the graph and add components. This object is the key to describe a structure. It is then very simple to create a creature that we have in mind (or for possible later use to generate them automaticly. . .). For instance, this is the code to create a snake with a given size : for the number of cell, we add a block and a joint.

```
def add_snake(m, size):
    #m is a Metastructure
    for i in range(size):
        m.add_block()
        m.follow_edge()
        m.add_joint()
        m.follow_edge()

    m.add_block()
```

Chapter 3

Control

Once the simulation is fully implemented, the goal is to use it as a testbench for different ways of controlling the structure. The second part of the project will be fully consecrated on this part. A few tools have been implemented to test some of the algorithms that are used in such problems.

3.1 PID Control of the joints

One of the problem to control accurately a joint is to adapt the command so that it can adapt to difficult situation. For instance, it is easier to walk in a swimmingpool than on the ground and this is why people recovering after an injury do aquatic training. For a joint, it can be easy to do a movement in the air, but the same movement is more difficult when touching the ground. One way to avoid this problem is to implement PID (Proportional Integral Derivative) controller. This kind of controller is used in a lot of different situation in the world of automation to control degrees of freedom. In or case, servomotors often integrate such loops to account for these changes of use. In the simulation, at each step, the command of each degree of freedom of the structure is calculated using such a PID controller.

One of the problem with the control of the joints is the unstability. On the simulation, as the angles have periodic values, if the joint has a high angle velocity, then between two steps, it can jump accross the boundary and start turning faster and faster as the PID controller will not be able to work as well as if the degree of freedom were a linear

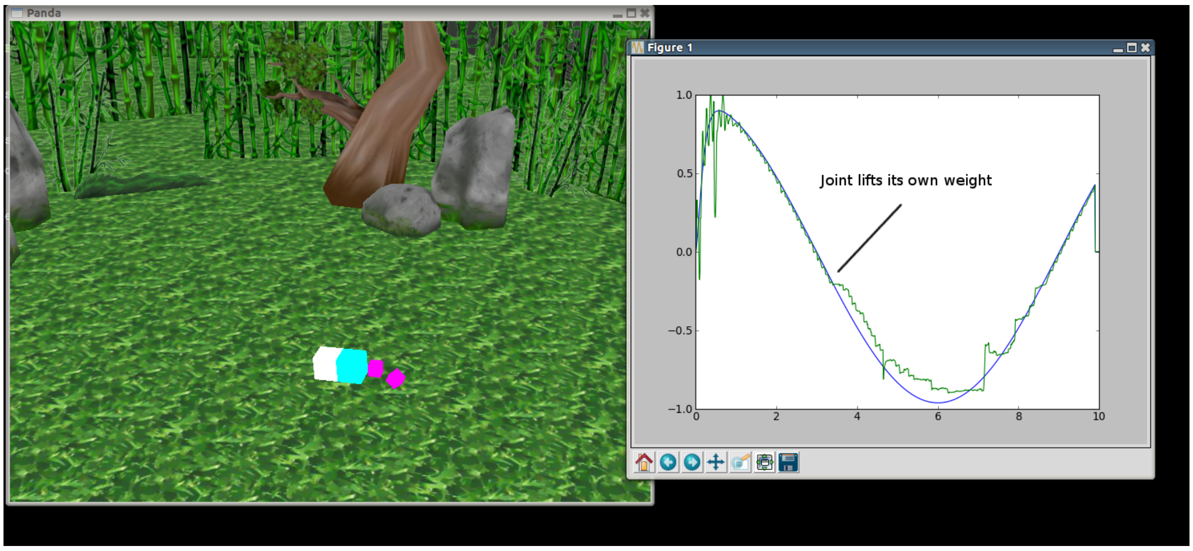

Figure 3.1: Answer of the joint to a sinusoid command with PID Control

parpameter. The problem with this situation is that it can actually be seen as a good result for the learning algorithm, simply because the fitness function is the velocity of the creature. There are several things to prevent from this situation :

- reduce the time between two steps of computation for the physics engine and the PID
- reduce the maximum torque available per joint
- punish the creature if the joints are moving to slow
- change the fitness function to maximize parameter under a certain energy level

Another issue is more a biological concern. The main goal of this project is to find solutions that are biologically inspired instead of using a classical automation approach (eventhough we can compare the elasticity of or muscles as such controllers…). Therefore though the use of PID is necessary in many robotics applications, we will try to avoid using the in the second part of the project.

3.2 Central Pattern Generator

Central Pattern Generators (CPGs) are neural networks, that can generate oscillation for the control of the muscles of or body. They are the consequence and the cause of the paradigm of periodic movement in the locomation of animals. Different models have been implemented to represent CPGs. We can represent them as a graph of coupled oscillator, where each node influence the behavior of its neighbours. For a first implementation I choose to test the model of CPG followed at EPFL ([3]).

The CPG Neural Network in that case, is a graph that follow the physical architecture of the robot, settting one node for each joint (hinge or vertebra) on the structure. The dynamic of the CPG is determined by a coupling weight matrix w_{ij}, a phase bias matrix between nodes φ_{ij}, the frequency of the different oscillator ω_i and the desired amplitude and offset of the oscillation. We can compute the angle using the following system of equation and an integration method (I used the Runge-Kutta method in this project)

$$\dot{\phi_i} = \omega_i + \sum w_{ij} * r_j * sin(\phi_i - \phi_j - \varphi_{ij}) \tag{1}$$

$$\theta_i = x_i + r_i * cos(\phi_i) \tag{2}$$

This two equation gives the angle of the oscillator (θ_i) depending on the state variable of a node: x_i, r_i, ϕ_i, that can be described respectively as the offset, the amplitude and the phase of the oscillator.

$$\acute{r_i} = ar(\frac{a}{r}(R_i - r_i) - \dot{r_i}) \tag{3}$$

$$\acute{x_i} = ax(\frac{a}{x}(X_i - x_i) - \dot{x_i}) \tag{4}$$

Equations (3) and (4) describe the dynamic of the amplitude and offset (a second order dynamic that converge to the desired values). This trick is to ensure continuity in the oscillations, even if some of the parameters of the oscillator change. a_r and a_x are gains to control the dynamic ($a_r = a_x = 20rad/s$ [3]).

A modification of this model is possible to plug the measured value of the degrees of freedom. Instead of using the second order control loop on θ_i which is achieved with the PID, we can set this control on the phase. Thatway, if the joint has troubles achieving his movement, for instance when hitting the ground, the phase will be modified and the perturbation will have an impact on othe joints through equation (1).

Oneway to do so is to add a term in the equation (1), with $\dot{\theta_{reali}}$ the mesured angle velocity of the joint.

$$\dot{\phi_i} = \omega_i + \sum w_{ij} * r_j * sin(\phi_i - \phi_j - \varphi_{ij}) + a_\phi * \frac{\dot{\theta_{reali}} - \dot{\theta_i}}{r_i * sin(\phi_i)} \tag{1}$$

If we derive (2) we get:

$$\dot{\theta_i} = \dot{x_i} + \dot{r_i} * cos(\phi_i) + r_i * sin(\phi_i) * \dot{\phi_i} \tag{2'}$$

By making the assumption that the dynamic of the amplitude and the offset is slow compared to the phase, we get

$$\dot{\theta_i} = r_i * sin(\phi_i) * \dot{\phi_i} \tag{2''}$$

Thatway, if we consider small variation of the phase, we can deduce an error term on $\dot{\phi_i}$ from the error on $\dot{\theta_i}$ given by $\frac{\dot{\theta_{reali}} - \dot{\theta_i}}{r_i * sin(\phi_i)}$ that we can control with a gain (a_ϕ). For example if the movement of a joint is made difficult because of the ground, then the mesured velocity of this joint will be smaller that expected. The consequence will be to accelerate the movement for this joint (the derivative of the phase will be bigger), but also for the other joints that are linked to this one. We can interprete this as neural communication in our body within the central pattern generator, but also as the elasticity between joints. For instance if achieving a movement is too difficult for a joint, using elasticity, it is possible to get help from joints that are near.

3.3 Liquid State Machine

ADD HERE

Chapter 4

Learning Methods

In this chapter we present different Learning Methods that have been tested and compared on this particular problem. All of these methods are optimization algorithms to find a minimum (or a maximum) over a set of parameters. It is not possible to get a gradient or Hessian matrix for a problem involving a simulation as fitness function. Therefore none of these algorithms use the derivatives of the function as information to find optima (unlike Gradient Method or Broyden-Fletcher-Goldfarb-Shanno (BFGS)). The following sections present different algorithms that can perform such a task. We then compare their results on the simulation. For coherence in this chapter, we use the following notation:

- $X = \{X_p, p \in \{1, ..., q\}\}$ is a set of q elements (or vectors) of the parameter space ($\mathbf{R}^n$), where n is the dimension of this space. $X_{p,i}$ is a scalar, and represent the i-th coordinate of the p-th element of the population.
- $f(\mathbf{R}^n \to \mathbf{R})$ is the fitness function we want to minimize.
- $X^{(p)}$ the p first element of the ordered list of element from set X where the order relation is given by $X_i \leq X_j$ iff $f(X_i) \leq f(X_j)$
- iterators p, q are over the population, where i, j are over the coordinate of each element

4.1 Genetic Algorithm

A Genetic Algorithm (GA) is an optimization algorithm that replicates the process of natural selection. GAs were introduced in the 60s and early 70s and are inspired from biology. They consist of a loop of four steps until convergence (or a proper solution) is reached : evaluation, selection, cross-over, mutation. A population X of p elements is randomly generated to initialize the algorithm. The evaluation step is just running the fitness function f over the new elements.

4.1.1 Selection

The first step is a selection step. The goal is to keep only a part of the population and eliminate the rest. This is directly related to Natural Selection, where the "weakest" animals are less likely to survive. They are different ways of keeping "good elements". The first idea that comes to mind in performing such a step is to keep the q best elements, ie $X^{(q)}$. The problem with this solution is that it prevent the population from "exploring" as all the best elements will be kept, it is more likely for the population to get stuck into a local minima. A widely use alternative is the Tournament Selection. We randomly select two elements and keep only the best of the two. We repeat this step until enough elements have been eliminated (usually half of the population). This alternative give a chance to all elements even if they are not in $X^{\{}q$ best.

Algorithm 1 Tournament Selection

```
$p = 0$
**while** $p < q/2$ **do**
  $X_1 = randomelement(X)$
  $X_2 = randomelement(X)$
  **if** $f(X_1 \leq X_2)$ **then**
    $X.eliminate(X_2)$
  **else**
    $X.eliminate(X_1)$
  **end if**
**end while**
```

4.1.2 Cross-Over

The cross-over is a step to repopulate after the selection. By taking two random elements of the set, we can compute new elements (usually 2) by performing a cross-over of the

the two parents. This step is inspired from reproduction in biology. The idea behind it is that by taking two good solutions, we can create 2 new elements that are likely to be good solutions. When the elements are scalar, a way of producing these solution is to use a barycenter of two solutions.

Algorithm 2 Cross-Over

```
for $p \in [1 : q/2]$ do
    $X_1 = randomelement(X)$
    $X_2 = randomelement(X)$
    $\lambda = randomfloat([0.5; 1.5])$
    $X.add(\lambda * X_1 + (1 - \lambda) * X_2)$
    $X.add(\lambda * X_2 + (1 - \lambda) * X_1)$
end for
```

4.1.3 Mutation

Finally the mutation step is a way of exploring new solutions. We randomly chose some elements to be mutated and modify them slightly. One way of doing the mutation when dealing with vector of real numbers is to use a gaussian distribution which use the previous elements as a mean, and a covariance matrix that depends of the population (for instance the covariance of p-nearest neighbors)

4.1.4 Discussion on the hyper parameters

4.2 Nelder-Mead method

The Nelder Mead method (or downhill simplex method) is a way of finding a local optimum, without knowing the gradient of the function in a given multidimensional space. We initialize a non-degenerated simplex in this space, then we follow this procedure (source: wikipedia):

- Ordering: we order the points of the simplex such that $f(x_0) >= f(x_1) >= f(x_2)\dot{>} = f(x_n)$, where f is the fitness function that we want to maximize (traveled distance...)
- We compute the center of gravity of all the points x_g
- We compute the reflection of x_n in respect to x_g ($x_r = x_g + (x_g - x_n)$)

- If $f(x_r) > f(x_{n-1})$ then we compute the expansion point : $x_e = x_g + 2 * (x_g - x_n)$ if $f(x_e) > f(x_r)$ we replace x_n with x_e else x_r and we go back to the first step.
- If $f(x_r) > f(x_{n-1})$ then we compute the contraction point : $x_c = x_g + 0.5 * (x_g - x_n)$ if $f(x_c) > f(x_n)$ we replace x_n with x_c and go back to the first step, else we do the next step
- a contraction homotetehia of center x_0 : we replace x_i with : $x_i = x_0 + 0.5 * (x_i - x_0)$ for $i > 0$ and go back to the first step.

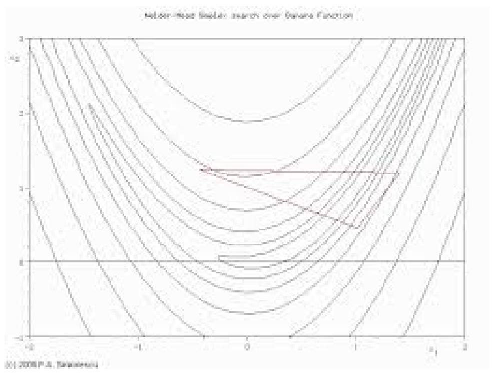

FIGURE 4.1: A simplex following the Nelder-Mead method

Moreover, in this problem the fitness function is not likely to be convex, as taking the mean of two good solutions for the movement of a structure does not necessary provide a good solution. The fitness function can also present some discontinuities or high variations because of collisions. Collisions are not continuous phenomena and even if the physics engine use smoothing techniques to simplify interractions. For instance there is a very small difference between a biped structure walking and a biped structure almost walking but with one leg that does not touch the ground, but the fitness function will give completely different results as one structure is moving and the other is not. Finally there is also the problem of consistency of a result, because two structures can behave differently for the same parameters, as the simulation can show chaotic behavior as small variations can have a big impact on the movement. For all these reasons, it is difficult to use classical optimisation for the creatures to learn how to move in the simulated environment.

It is then possible to learn the parameters of the CPG to optimize the movement of the structure. Instead of having to find the value of the angles, the CPGs act as basis function for the angles, and thatway we reduce the space of research to a space with finite dimensions. All the parameters (frequency, offset and amplitude of each node) are scaled to fit between 0 and 1. A simple fitness function can be extracted from the simulation, for example the distance traveled by the head of the structure.

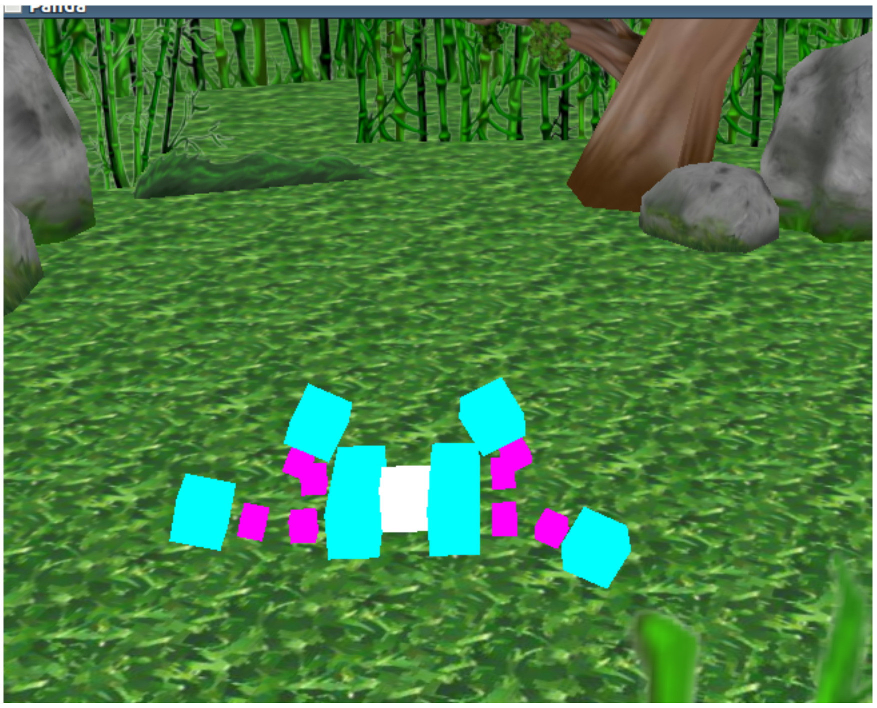

FIGURE 4.2: A structure with four legs learning

4.3 Random Search

A first very simple way of finding good solution is random search. We test a random set of parameters and keep the vector of parameters showing the best results. This has the benefit of being very simple to implement and provides a good testbench for fixing issues with the simulation. A first problem observed was the consistency of the solution, testing the same parameters can lead to very different results. A first way to correct this was to take longer sample (about 20 sec of simulation). Even with this correction, online learning brought some issues, as it is possible that a good results is only good because of the initial configuration given by testing previous movement. A way to correct this is to set all angles to a default value and wait for the structure to be have a null velocity. This also led to some issues and showed solution using the first movement to jump as far as poissible.

4.4 Simulated Annealing

This method gave some good results for simple creature, (without to many degrees of freedom). I modified it to evaluate the function again, each time we sort the points of the simplex. Though it is less efficient, thatway, it is possible to prevent from having a lucky trial. For instance, as the test depends of initial condition, sometimes a result can be really good, but cannot be repeated, for instance a four legged structure can get a good score but end up on the back after one trial and will not be efficient on the next ones.

Chapter 5

Results

5.1 Simulation reproductability

ADD HERE -¿ discussion on the on the reproductability of the simulation -¿ graph showing noise of the results (all the scores for constant parameters)

5.2 Creatures taking advantages of simulation bugs

-¿ discussion -¿ link to video

5.3 comparaison

-¿ graph for each couple (model, learning method)

Chapter 6

Conclusion

ADD HERE

Bibliography

[1] Karl Sims. Evolving 3d morphology and behavior by competition. *Artificial life*, 1 (4):353–372, 1994.

[2] D. Marbach and A.J. Ijspeert. Online optimization of modular robot locomotion. In *Mechatronics and Automation, 2005 IEEE International Conference*, volume 1, pages 248–253. IEEE, 2005.

[3] Alexander Sproewitz, Rico Moeckel, Jérôme Maye, and Auke Jan Ijspeert. Learning to move in modular robots using central pattern generators and online optimization. *The International Journal of Robotics Research*, 27(3-4):423–443, 2008.

[4] Thomas Geijtenbeek, Michiel van de Panne, and A. Frank van der Stappen. Flexible muscle-based locomotion for bipedal creatures. *ACM Transactions on Graphics*, 32 (6), 2013.

[5] Russell Smith et al. Open dynamics engine, 2005.

[6] John Romanishin. Mit's m blocks toward modular robotics. "http://web.mit.edu/newsoffice/2013/simple-scheme-for-self-assembling-robots-1004.html", 2013.

[7] Christoph Lang. *Panda3D 1.7 Game Developer's Cookbook*. Packt Publishing, 2011.

[8] Herbert Jaeger. Echo state network. *Scholarpedia*, 2(9):2330, 2007.