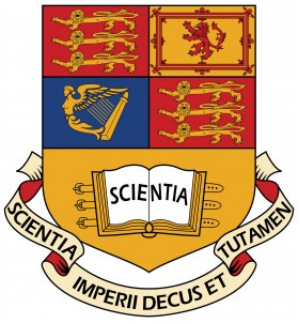
Imperial College of London

Research Project Report

Learning walking skills and Laws of Commands for modular Robots

Author:
Clement Jambou

Supervisor:
Pr. Murray Shanahan



January 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

Though classical automation has proven to be really efficient for many problems in the last years so that we see consequence of its use everyday in our life, non-linear problems and problems with many degrees of freedom seem to be resistant to the classical approach. The goal of this project is to generate automaticly laws of command for complex modular structures composed of simple blocks and joints. By learning from a 3D world under simulated physics laws the robot will learn to move on the ground by itself. It will then be possible for someone to control it via a simple interface like a joystick. Another goal/demo is to start in a situation with blocks and joint scattered in the 3D world and find the best way to regroup all of them in one structure using the pre-generated command laws/results for all intermediary structures.

Contents

List of Figures

List of Tables

Chapter 1

State of the Art

The first remarquable examples of modular robotics learning to evolve in a 3D simulated world is due to Karl's Creature in 1994 [**?**]. Since then Genetic Algorithms have proven to be very efficient to solve motions problem especially in trying to maximize on parameter (speed for instance) [**?**] [**?**] [**?**]. Some of those learning algorithm are also tested on physical implementation of complex structures [**?**] [**?**]. More recently new techniques using reinforcment learning [**?**] showed interesting results. The M-blocks initiative at MIT [**?**] tackles the mechanical engineering chalenges behind modular robotics, but a lot of challenges remain from a software perspective in order to use modular robotics in the future fields we expect it to have an impact one, such as space (or non-friendly human environments) exploration and construction, medicine, ...

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 this two phenomena on complex structure can lead to situations 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` `ODE` (`Open Dynamics Engine`). 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 all the structure on different surfaces, (not necessarly plans). Also in order to simplify the structure, so that the modules are simple shapes, all the robots are only composed of cubes, linked by different type of joints. 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 value and a command to the motor.

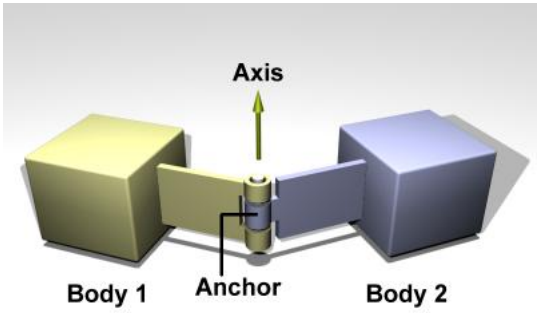


FIGURE 2.1: A 3D representation of a Hinge Constrain

2.2 Visual rendering

A good thing about the physical engine is that it is completly separated from the rendering part. That way, it is possible to run much faster for 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, that integrate both a rendering librairy using OpenGl and different physics engines (ODE and bullet). All this tools are developped in C/C++ with binding for python, which makes it an easy tool to create animation movie, games or simulations.

2.2.1 The coordinate system

In order to represent all the objects that we manipulate in the simulation (physics and rendering), panda3d 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. 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

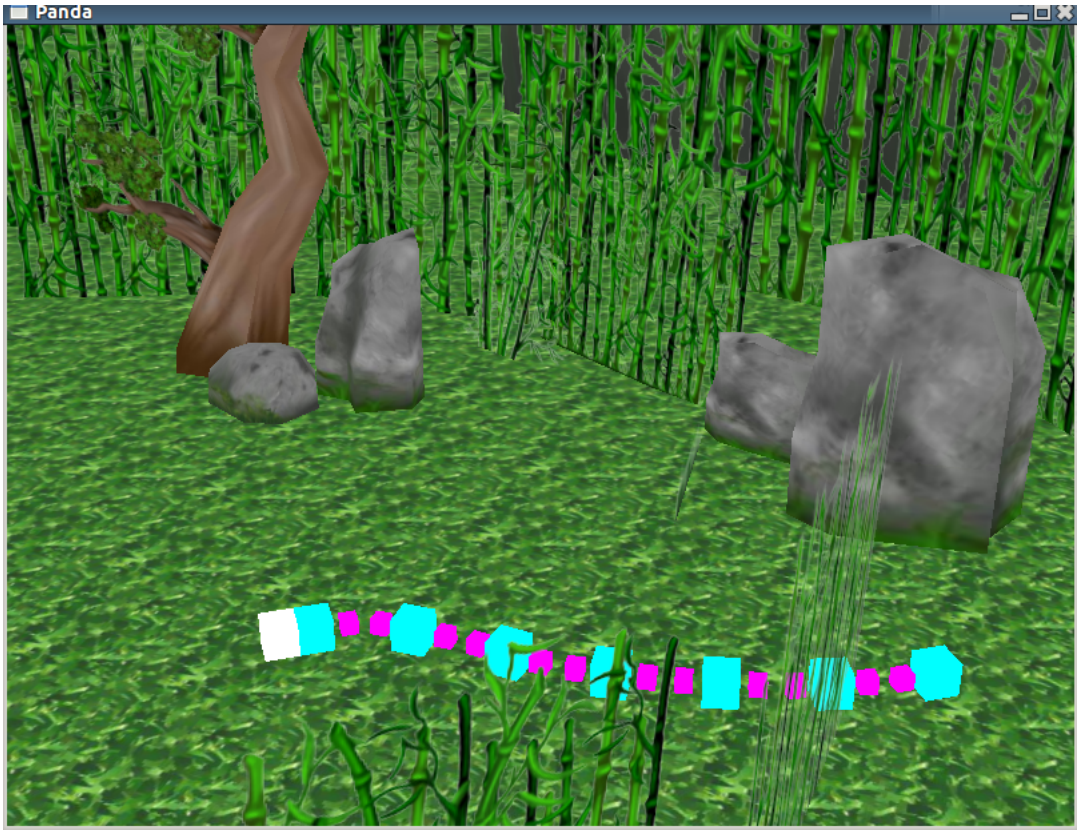


FIGURE 2.2: 3D Rendering of a snake in the environment

($\mathbb{R}^3 \to \mathbb{R}^3$) that can be interpreted as the set of combined rotations and homotetia, where the matrix multiplication is the composition of such transformations, and a translation. Manipulating translations requires to use 4 by 4 matrices instead of 3 by 3 but the properties of the product are kept, which makes it a very useful tool to manipulate 3d objects. For instance, if the children of the node is 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.

Picture of the 4*4 Matrix

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 :

- 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.

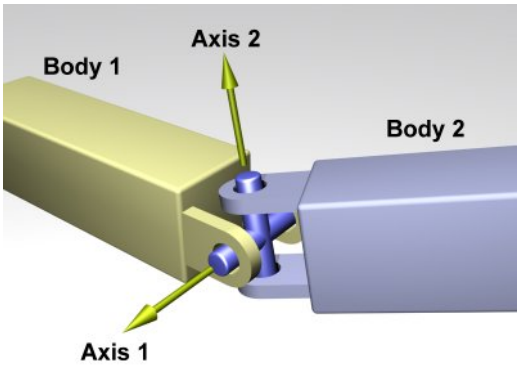


FIGURE 2.3: A 3D representation of a Vertebra Constrain

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 structure 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()
```

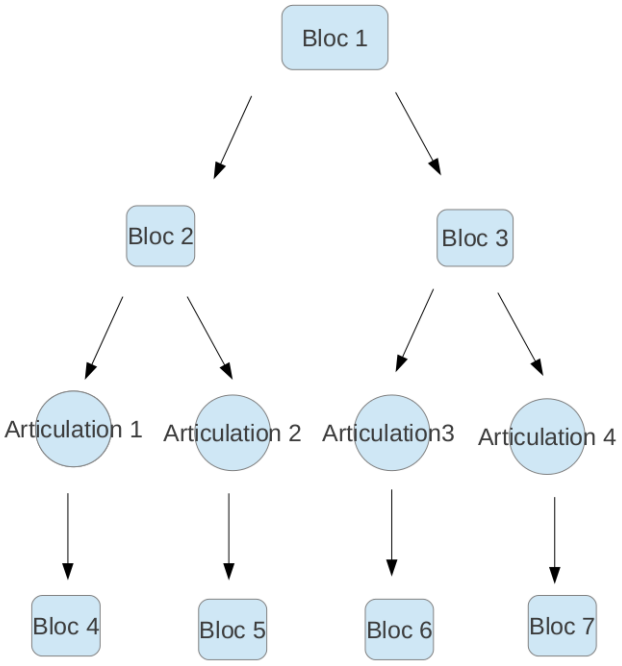


FIGURE 2.4: Structure described as a graph

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.

A few issues come from introducing PID in this problem. The stability is one of them. The PID loops need to be fast enough to be stable, which means that reducing the time between two steps of calculation on the physical world. The other problem is more a biological concern. The main goal of this project is to find solutions that are biologically

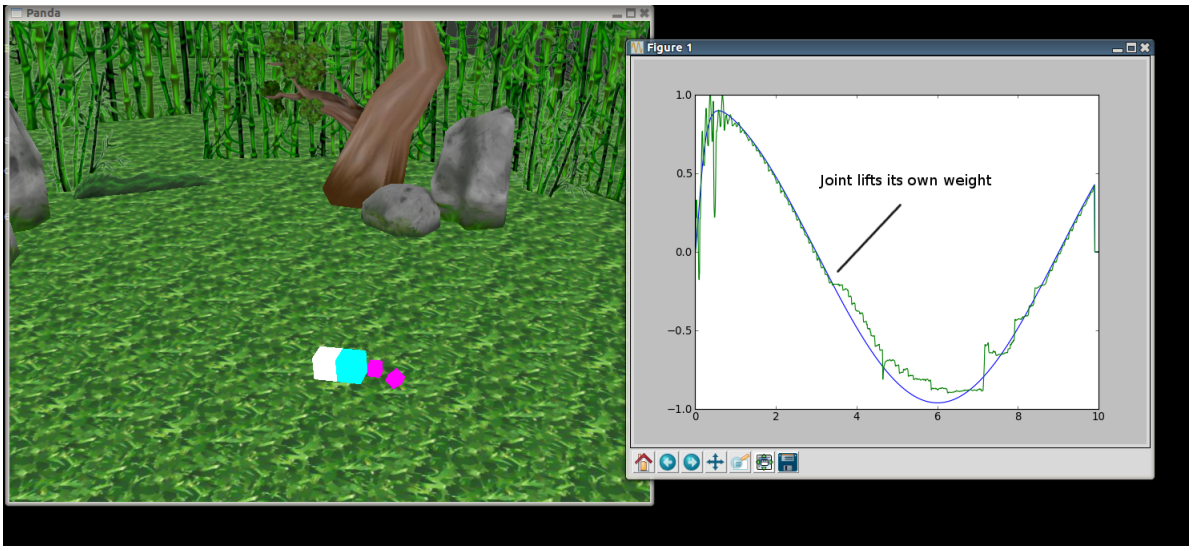


FIGURE 3.1: Answer of the joint to a sinusoid command with PID Control

inspired instead of using a classical automation approach. Therefore though the use of PID is necessary in many robotics applications, we will try to avoid using them in generating oscillations.

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 (INCLUDE CITATION). 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}, setting 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)

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

$$\theta_i = x_i + r_i * cos(\phi_i) \quad (2)$$

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.

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

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

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 ($a_r = a_x = 20rad/s$ CITATION).

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.

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

If we derive (2) we get:

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

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

$$\dot{\theta}_i = r_i * sin(\phi_i) * \dot{\phi}_i \qquad (2")$$

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.

Chapter 4

Learning

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. It is not possible to get a gradient or Hessian matrix for this problem, as the function comes from the simulation. Moreover, 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 are not continuous phenomenon and even if the physics engine use smoothing techniques to simplify interractions, 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.

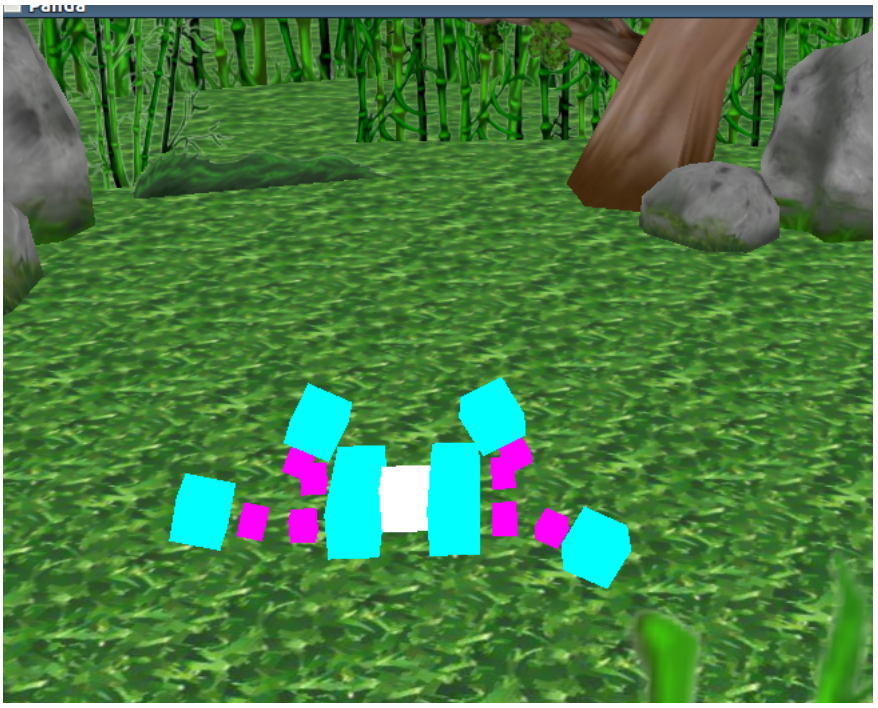


FIGURE 4.1: A structure with four legs learning

4.1 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.

4.2 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 homotethia 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.

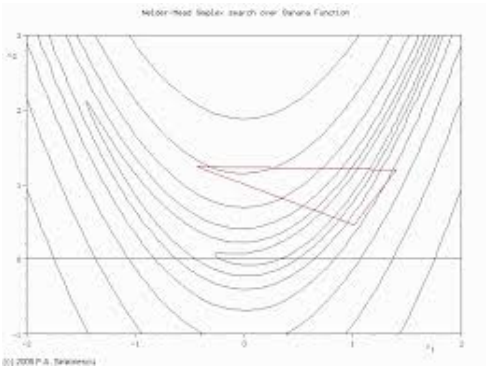


FIGURE 4.2: A simplex following the Nelder-Mead method

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.