

Modeling of a real quadruped robot using Webots™ simulation platform

Jean-Christophe Fillion-Robin¹
 School of Computer and Communication Sciences (I&C)
 Ecole Polytechnique Fédérale de Lausanne
 CH-1015 Lausanne, Switzerland

Abstract — The fundamental achievement of this project was the development of a three-dimensional model that closely resembles the real quadruped robot. While the static characteristics of the model were validated against the actual ones of the real robot, the dynamical ones were defined by the results of applying the inverse kinematics model. These were considered best when compared to the ones yielded by the particle swarm optimization (PSO) of central pattern generator (CPG), an alternate method to determine the properties in question. The development of this model will serve as the foundation of a demonstration involving four to six robots playing on a virtual soccer field.

This paper highlights and contextualizes our implementation and outlines the crux of the problem in order to motivate future related research.

Index Terms— Quadruped robot, 3D model, simulation, four-legged walking gait, central patterns generators, particle swarm optimization, inverse kinematics

I. INTRODUCTION & MOTIVATION

The modeling of a real robot is a complex and passionating challenge. On the crossing point of mechanics, physics and computer-science, the development of a complete model involves multiple tasks ranging from the 3D modeling of the different body parts, the measure of the different physic properties, the understanding of Webots™ simulator to the development of a central pattern generator or inverse kinematic model allowing the robot to move.

The project was built on the top of two cornerstones: the elaboration of an accurate model both efficient and realistic, and the development of a demo showing the capacities of Webots™ simulator to render a 3D model while simulating the physics.

Multiple component were developed allowing to ease the coding of the model, to validate easily the static characteristics, to distribute the optimization of a central pattern generator on cluster of computers.

Experimental results were gathered leading to clear conclusions. However, a additional test would validate the conclusions even further. There is still a large amount of work to be done and, with this document in hand any master-level

engineering student should be able to take the project further. Despite of what is outstanding, the overall satisfaction at this point in time from both, team member and collaborators, is high.

The remaining of this document aims at describing the Bioloid kit used to build the robot, presenting the Webots™ simulator used to render and simulate the model, and, finally illustrating the different phases of the project.

II. BIOLOID KIT

Bioloid Kit is a composed of a collection of block-shaped parts (see Appendix) and servos (see II.C), a sensor module(see II.B) and a control unit (see II.A) that the user can assemble together to build a sophisticated robots. The connection between the different parts can be easily achieved using a simple screw driver, the frames made from injection molded plastic fit and connect perfectly (see Figure 1).

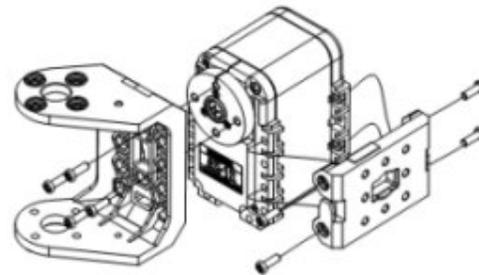


Figure 1: Example of connection of a AX12 dynamixel servo

A. Control unit and Communication bus

The control unit is so called the CM5 module and is based on a micro controller Atmel ATmega 128 [4]. The rechargeable battery pack of 9.6V is also embedded in the CM5 module. All servos have a network ID programmed in their non-volatile memory and are connected together via a TTL Serial Network (see Figure 2).

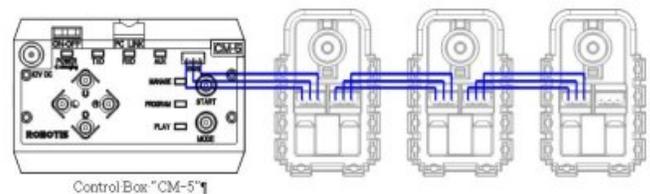


Figure 2: Servos and sensors connected to CM5 unit via 3 wire TTL serial network

¹Report written July 04, 2007.

Each unit in the Bioloid Kit has his own assigned ID [5]. The CM-5 has ID200. The sensor module has ID100. The servos module can have IDs between 1 and 19.

Figure 3 illustrates the connection of the different module of the real robot and the corresponding module IDs.

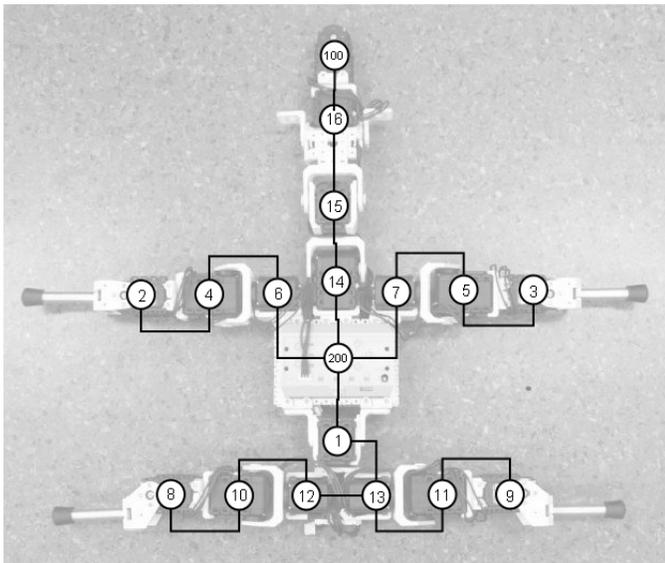


Figure 3: Wiring of the Dynamixel module

B. Sensor module

The Sensor Module AX-S1 is a Smart Sensor Module that integrates the following functions [21]:

- embedded with three directions infrared sensor, making it possible to detect left/center/right distance angle as well as the light.
- built-in remote control sensor in center, making it possible to transmit and receive infrared data between sensor modules.
- built-in micro internal microphone, making it possible not only to detect current sound level and maximum loudness but also an ability to count the number of sounds, for instance, the numbers of hand clapping.
- Built-in buzzer allows the playback of musical notes and other special note effects.

Weight	37g
Resolution	10bit (1024)
Voltage	7V~10V (Recommended voltage: 9.6V)
Supply Current	40mA
Operate Temperature	-5°C ~ +85°C
Command Signal	Digital Packet
Protocol Type	Half duplex Asynchronous Serial Communication (8bit,1stop,No Parity)
Link (Physical)	TTL Level Multi Drop (daisy chain type Connector)
ID	254 ID (0~253)
Communication Speed	7343bps ~ 1 Mbps
Feedback	Infra-red Sensor, Internal Mic, Temperature, Input Voltage, IR Remocon Tx/Rx Data, etc.
Material	Engineering Plastic

Table 1: AX-S1 sensor module technical specifications [21]

C. AX12 module

The AX12 module [22] is a smart, modular actuator that incorporates a gear reducer, a precision DC motor and a control circuitry with networking functionality, all in a single package. Despite its compact size, it can produce high torque. It also has the ability to detect and act upon internal conditions such as changes in internal temperature or supply voltage. Position and speed can be controlled with a resolution of 1024 steps (see Figure 4).

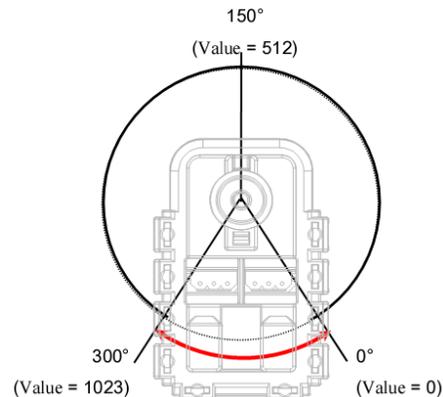


Figure 4: Rear view of the Dynamixel AX-12 module and the corresponding actuation position.

The AX12 module[22] can alert the user when parameters deviate from user defined ranges (e.g. internal temperature, torque, voltage, etc) and can also handle the problem automatically (e.g. torque off).

		AX-12	
Weight (g)	55		
Gear Reduction Ratio	1/254		
Input Voltage (V)	at 7V	at 10V	
Final Max Holding Torque(kgf.cm)	12	16.5	
Sec/60degree	0.269	0.196	

Resolution	0.35°
Operating Angle	300°, Endless Turn
Voltage	7V~10V (Recommended voltage: 9.6V)
Max. Current	900mA
Operate Temperature	-5°C ~ +85°C
Command Signal	Digital Packet
Protocol Type	Half duplex Asynchronous Serial Communication (8bit,1stop,No Parity)
Link (Physical)	TTL Level Multi Drop (daisy chain type Connector)
ID	254 ID (0~253)
Communication Speed	7343bps ~ 1 Mbps
Feedback	Position, Temperature, Load, Input Voltage, etc.
Material	Engineering Plastic

Table 2: AX-12 module technical specifications [22]

D. Behavior editor

The Behavior Control Program allows to define a set of rules. Then, as a function of the robot current state and the outcome resulting in the application of these predefined rules, actions are done.

The description of the different rules is done using a graphic based programming environment (see Figure 5), in this way all

motion defined using the Motion editor (see II.E) can be connected together depending on input from the sensors and programmatic logic.

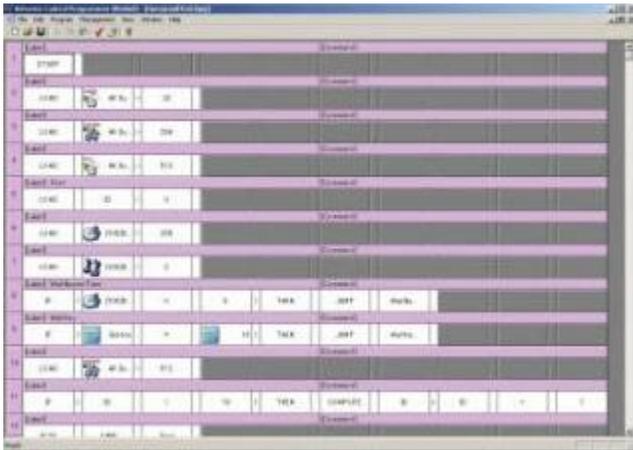


Figure 5: Behavior editor GUI

The commands provided with the Behavior Control Program include:

- program control commands (START, END),
- conditional branching commands (IF,ELSE IF,ELSE,CONT IF) with conditional operations ($=$, $>$, and $>=$, $<$, and $<=$),
- program sequencing commands (JUMP & CALL/RETURN),
- numeric commands (COMPUTE), and
- assignment commands (LOAD).

After a Behavior control program is defined, this one can be tested remotely or uploaded to the CM5 unit.

E. Motion editor

The Motion Editor is a package that allows the user, using a graphics based interface (see Figure 6), to move the motors of a robot simply by increasing or decreasing the number that describes the motors current position.

Motions are built up frame-by-frame - very similar to a story board in an animation sequence. This allows quite complicated "animations" to be programmed and tested. Once a motion has been defined it can then be downloaded into the CM5 memory and called from the Behavior Control Program.

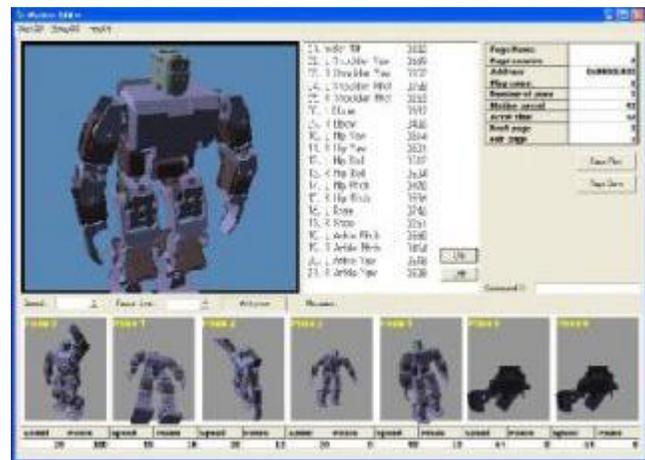


Figure 6: Motion editor GUI

There is two way of recoding the frame:

- According to a function defined by the user, all servo positions can be pre-computed and entered manually in the motion editor. This method presents the advantages of being rigorous since all servo positions are set considering the shape and the geometry of the robot.
- Since the servos can be aware of their own position, the current position of the robot can be recorded and a sequence can be described step by step. The cons of such a method is, for example, concerning the quadruped robot, the in-accuracy between the position recorded for each legs. The position of the members being set manually, plus the fact the robot can fall due to his proper weight, makes this method a non rigorous way to implement gait or motion. Nevertheless, this later one allows to fast prototyping motion and could help to check ideas.

The motion editor is a valuable option if the user limits himself to program the existing example. When a new robot is design, since the 3D model is not existing in the motion editor, the user isn't able to visualize the frames already entered. He can only visualize the current position being reproduced remotely on the real robot while entering the frame.

III. WEBOT™ SIMULATOR

Webots™ is a commercial software developed by Cyberbotics Ltd. It is “a mobile robotics simulation software that provides you with a rapid prototyping environment for modeling, programming and simulating mobile robots. The provided robot libraries enable you to transfer your control programs to several commercially available real mobile robots. Webots™ enable you to transfer your control programs to several commercially available real mobile robots. Webots™ lets you define and modify a complete mobile robotics setup, even several different robots sharing the same environment. For each object, you can define a number of properties, such as shape, color, texture, mass, friction, etc. You can equip each robot with a large number of available sensors and actuators. You can program these robots using your favorite development environment, simulate them and optionally transfer the resulting programs onto your real robots. Webots™ has been developed in collaboration with the Swiss

Federal Institute of Technology in Lausanne, thoroughly tested, well documented and continuously maintained for over 7 years.”[1]

The core of Webots™ is based on the robust and powerful physics engine: Open Dynamics Engine (ODE)[2]. The two main components of ODE are the rigid body dynamics simulation engine and the collision detection engine. As we will see, the development of a model involved multiple phases: ranging from the “drawing” of the body parts (see IV.A), the definition of the bounding objects (see IV.C.3), the setup of the physical properties (see IV.C.2). ODE is used both as a research-helper package or as a computer game development library helping at providing more realistic behaviors. Open-source and freely available, as of today, ODE is involved in the development of almost 100 referenced projects[3]. Started in 2001 and with the huge number of contributors, as a mature C++ API and a platform independent package, ODE is known to be a successful and promising Physics engine.

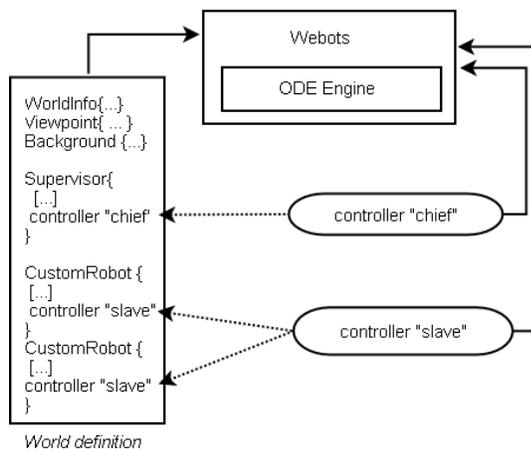


Figure 7: Webots™ functional diagram

The basic behind the design of a Webots™ model are quite simple. A fully functional model (see Figure 7) is composed of at least a world file (see III.2) and a controller. A controller is a program mapped to an entity represented in the world, allowing to give it instruction regarding his surrounding environment, his internal state, instruction received from an other controller [...].

The execution of world starts with the opening of a world file (WBT file extension). Webots™ parses the file and check for inconsistencies. Then, it extracts the controller(s) name(s) corresponding to the different entities (customRobot, Supervisor node). Finally, each entity has his own instance of controller running. Webots™ with the help of ODE engine can animate and render the scene in respect of the physics. The specificities of the simulator regarding the development of the model will be reviewed in the subsidiary parts of this paper.

IV. MODELING

As stated in [1], the design of a model out of one's imagination could be done in few hours. In our case, it's a quite more

sophisticated problem since we have the real robot and we want to build its most accurate representation both in respect of its visual and its body dynamics.

A. Visual Aspect

The Bioloid kit comes with a set of example assembly instructions. For each of them, a motion file and behavior file are provided. Since the motion file contains the 3D model with the exact shape of all the body parts, it could be interesting to be able to extract the useful informations and generate one file for each ones. A collaborator, Laurent Lessieux, specialist of robotic modeling and simulation, provided me with some code able to create OBJ files for each part from the X file provided with the Bioloid Kit. The X files are the 3D description files used by Microsoft DirectX API[23]. Indeed, there is already a set of examples provided with the kit and each example has the corresponding 3D model allowing to visualize it in the motion editor. Thanks to his work, the initial phase involving the shaping of the 3D body parts was speed up.

1) Shape simplification using 3D modeling tool

Having all 3D files corresponding to all the different shapes is a good advantage. The OBJ format corresponds to the “standard 3D object file format created for use with Wavefront's Advanced Visualizer. Object Files are text based files supporting both polygonal and free-form geometry (curves and surfaces).”[6][7]

It exists an open-source 3D modeling and rendering studio allowing to import an object file and export it to POV[8] file format or VRML(Virtual Reality Modeling Language) file format. Called Art of Illusion (AOI)[9], this Java-based tool provides an easy way to model 3D shape or even edit existing one.

VRML is a standard file format to represent 3D interactive graphic vectors.[10]

VRML language allows to natively describe spheres, cubes, cones or extrusions. If the modeled element is converted to a triangle mesh shape. The efficiency of the rendering will be a direct function of the number of triangle meshes.

Some of the shape provided were too complex, they presented a high degree of details and for the sack of the modeling, it was not necessary to keep such low level details. (see Figure 8 and Figure 9)

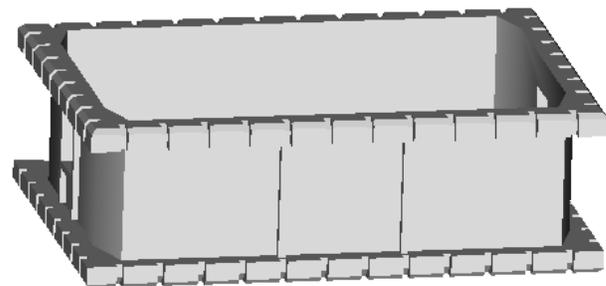


Figure 8: CM5 shape before simplification - the VRML exported file is about 158.5Kb

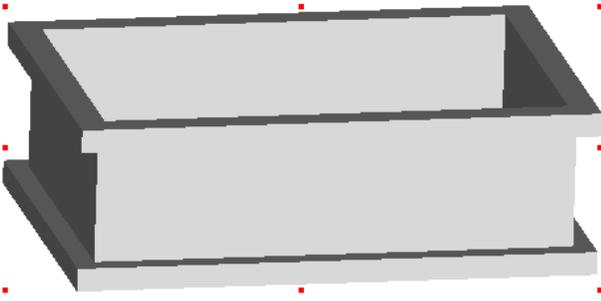


Figure 9: CM5 shape after simplification - the VRML exported file is about 5.5Kb

The 3D modeling of a model is a trade of between visual realism and rendering complexity.

2) Webots™ scene description

The description of scenes follows a VRML-like language, in that way all shapes modeled using the 3D modeling tool mentioned above can be easily imported.

There is two ways of describing a world scene, either using the Webots™ built-in VRML editor or using an external editor. Kdevelop[24] revealed to be an interesting option since it can handle VRML file easily and fold/unfold VRML node. This last feature of the editor allows to get a better overview of the work done so far.

The Webots™ Nodes chart (see Appendix) is useful to understand what are the existing nodes, and from what kind.

3) Model of the quadruped robot

The elaboration of the quadruped robot model was a tedious and long process. While at it, I tried to keep the model clear, efficient and simple to understand. The various part used to construct the robot are referenced in the QuickStart manual[25] with a letter 'F' and a number discriminating the part. The same nomenclature is used in the world file. The Table 13, available in the Appendix, shows their names and their corresponding shapes.

A given shape is oriented in the world, then depending on their location in the node hierarchy, this one needs to be rotated. Within VRML, rotation are expressed with the notation “euler angles/axis”[26] (the 3D coordinate of one point and an angle θ) the field *rotation* allows to apply such a rotation. The common way of expressing rotation and visualizing easily this one is the representation “euler angle” (an angle of rotation corresponding to each axis). To ease the development of the model, a set of two Matlab functions allows the conversion between the two representations (see Appendix).

Figure 10 shows the final quadruped robot model rendered in Webots™. A simplified and more readable release of the world file is also available in Appendix. The complete listing will be accessible from the project web page.

Following the recent mails and over skype exchanges with the members team of Newcastle University, responsible for the hardware design of the robot, they updated the design of the

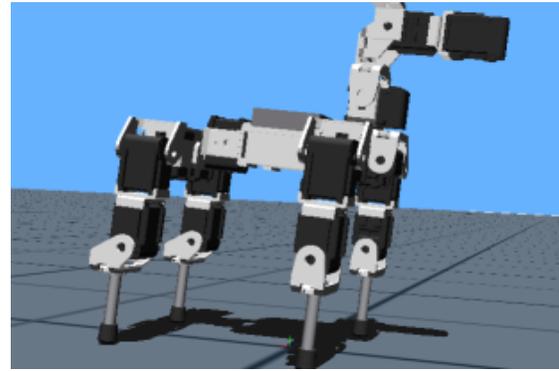


Figure 10: Model of the quadruped robot rendered in Webots™

robot and added wedges below the “knee” actuator to add a permanent offset of 18 degrees (Figure 11). The purpose of this update was to make the robot able to walk with an aibo-style and ease the development of an efficient gait. In the framework of this project and due to time constraints, only the initial design of the robot has been used and validated.

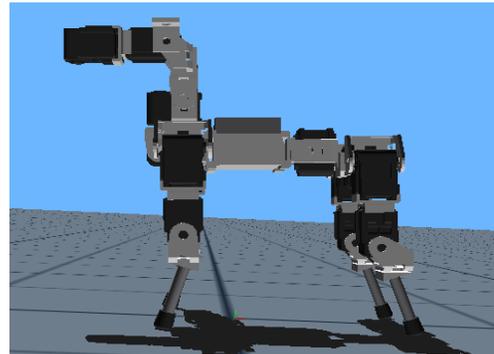


Figure 11: Model of the updated quadruped robot rendered in Webots™, having the 18 degrees wedges

B. Animate the model

The servo nodes(see Figure 12) are the links between the controller and the visual representation of the model. Giving orders to these servos, the Webots™ rendering engine is able to recompute the new position of the different joints, this way the model is animated and can interact with the physics of the world.

Table 3 summarizes the measures. The robot presenting similar body part, it's has been admitted their weight was equals. That hypothesis explains the redundancy of the value.

b) Center of Mass

The next step was to determine the position of center of mass (CoM) of the different body parts. The letter scale used before wasn't able to provide the right information concerning the X, Y, and Z location of the CoM. A simple way to overcome that problem was to construct a CoM measurer.

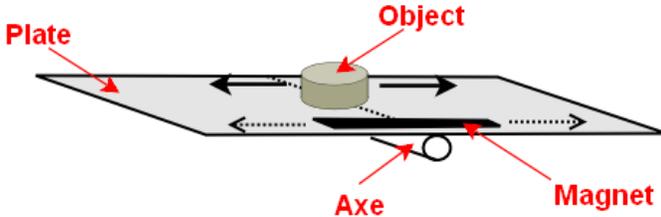


Figure 14: Functional schematic of the CoM measurer

As illustrated on Figure 14, the CoM measurer is composed of 3 main elements: the plate, the magnet and the axes. Before using it, the first step is to make the “zero”. To achieve that purpose, the magnet can be slid from left to right or from right to left until the plate is almost at equilibrium. The second phase concerns the measuring of the CoM itself. The body part is set on the plate and smoothly moved from left to right or right to left until the equilibrium state is reached. That last step should be executed for the 3 axis(X, Y, Z) of the body part. Particular attention has to be observed while the body part is set on the plate, indeed, the 3 axis should match as much as possible the one of the modeled shape. That was, the measured CoM can be directly reported in the model definition.

In order to measure the position of the CoM, while the body part is disposed on the plate and the equilibrium state is reached (or the closer possible to), a pencil can be used to mark the position of a characteristic point of the body part. Afterward, with a ruler, it's easier to measure the effective distance of the marked point to the central axis of the measurer.

After the measure is done, it's important to translate it into the coordinate system of the servo node (see IV.C.2.g, centerOfMass).

The shape of the body parts should also be considered, and when it's possible, for example in case of symmetry relative to the servo axis and assuming the internal weight distribution is uniform or equally distributed, the CoM value corresponding to the axis X, Y or Z should be 0.

2) Physics applied to Webots™

Within Webots™, the physical characteristics of any node derived from the *solid* node can be described adding the *Physics* node (see Figure 15) which allows to define a given number of physics parameters to be used by the physics simulation engine.

```
Physics {
  SFFloat density 1000 # (kg/m³) if -1 use mass
  SFFloat mass -1 # (kg) ignored if density!=-1
  MFFloat inertiaMatrix [] # 9 float values
  SFFloat bounce 0.5 # [0,1]
  SFFloat bounceVelocity 0.01 # (m/s)
  SFFloat coulombFriction 1 # see ODE documentation
  SFFloat forceDependentSlip 0 # see ODE documentation
  SFVec3f centerOfMass 0 0 0 # (-inf,inf)
  SFRotation orientation 0 1 0 0 # of the inertia matrix
}
```

Figure 15: Webots™ specification of the Physics node

Accurate informations about the different physics parameters are given in the following subsections.

a) density or mass

It “can be used to define the total mass of the solid. If the *density* field is set different from -1, then it is used regardless of the *mass* field to compute the mass of the solid object. Otherwise, the *mass* field, which should be set to a positive value, is used. You should never set both the *mass* and the *density* to -1, otherwise the results will be undefined. Rather it is highly recommended to set either the mass or density to -1 and the other field should be set to a positive value. If the *density* field is a positive value and the *mass* field is set to -1, the actual mass of the Solid node will be computed based on the specified density and the volume defined in the boundingObject(see IV.C.3) of the Solid node. However, this computed mass will not be displayed in the mass field which will remain -1.”[12]

b) inertiaMatrix

It “defines the inertia matrix as specified by ODE. If this parameter is empty or contains less or more than nine floating point values, it is ignored. Moreover, if the mass field is -1, the *inertiaMatrix* field is ignored. If it contains exactly nine floating point values, and if the mass field is different from -1, then it is used as follow: the nine parameters are the same as the ones used by the *dMassSetParameters* ODE function. The parameters given in the *inertiaMatrix(1)* are: cg_x , cg_y , cg_z , I_{11} , I_{22} , I_{33} , I_{12} , I_{13} , I_{23} , where (cg_x, cg_y, cg_z) is the center of gravity position in the body frame. The I_{xx} values are the elements of the inertia matrix, expressed in $kg.m^2$ ”[12]

$$inertiaMatrix = \begin{pmatrix} I_{11} & I_{12} & I_{13} \\ cg_x & I_{22} & I_{23} \\ cg_y & cg_z & I_{33} \end{pmatrix} \quad (1)$$

c) bounce

It “defines the bounciness of a solid. This restitution parameter is a floating point value ranging from 0 to 1. 0 means that the surfaces are not bouncy at all, 1 is maximum bounciness. When two solids hit each other, the resulting bounciness is the average of the bounce parameter of each solid. If a solid has no Physics node, and hence no bounce field defined, the bounce field of the other solid is used. The same principle also applies for to *bounceVelocity*, *staticFriction* and *kineticFriction* fields.”[12]

d) *bounceVelocity*

It “defines the minimum incoming velocity necessary for bounce. Incoming velocities below this will effectively have a bounce parameter of 0.”[12]

e) *coulombFriction*

It “defines the friction parameter which applies to the solid regardless of its velocity. Friction approximation in ODE relies on the Coulomb friction model and is documented in the ODE documentation. It ranges from 0 to infinity. Setting the *coulombFriction* to -1 means infinity.”[12]

f) *forceDependantSlip*

It “defines the force-dependent-slip (FDS) for friction, as explained in the ODE documentation. FDS is an effect that causes the contacting surfaces to slide past each other with a velocity that is proportional to the force that is being applied tangentially to that surface. It is especially useful to combine FDS with an infinite *coulombFriction* parameter.”[12]

g) *centerOfMass*

It “defines the position of the center of mass of the solid. It is expressed in meters in the relative coordinate system of the Solid node. It is affected by the orientation field as well.”[12] The measure of the CoM X, Y or Z location is relative to a side of the body part or an other noticeable point, before using it in the physics node definition, One's should take care of translating that value into the servo coordinate system.

h) *orientation*

It “defines the orientation of the local coordinate system in which the position of the center of mass (*centerOfMass*) and the inertia matrix (*inertiaMatrix*) are defined.”[12]

3) Bounding objects

Since the modeled shapes are complicated triangle meshes, these one can't be used by the collision detection engine. It will result in too complex computation and the overall simulation will be slow down. An approximation of the bounding object should be done. There is no automatic process to determine there bounding object. According to the specificities and the purpose of the simulation, the user should define them using simple object like “sphere”, “box” or “cylinder”. Within Webots™, when the user click on the robot model (Figure 16), the bounding object are highlighted.

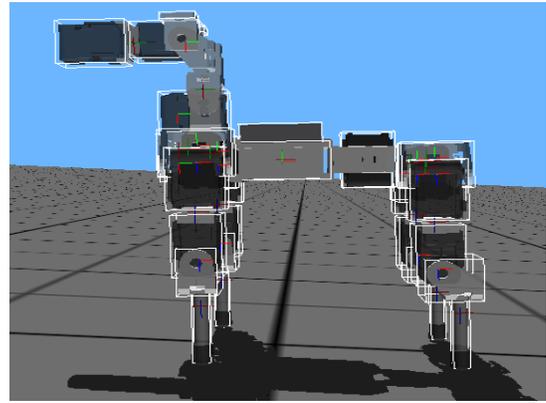


Figure 16: Robot model rendered in Webots™ with his bounding objects highlighted

Particular attention should be consider while defining the bounding object of the crutches. Indeed, the crutches are the contact point of the robot with the ground. This one was approximated using a box and a sphere (see Figure 17).

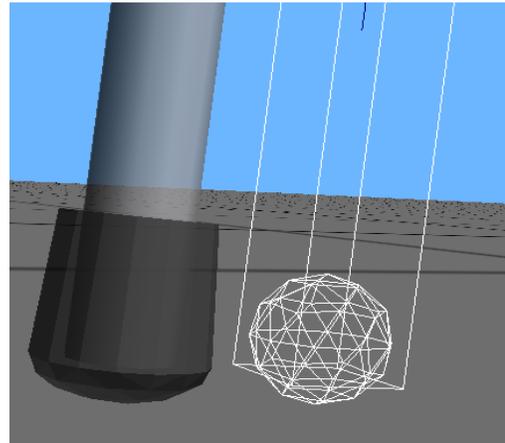


Figure 17: Approximation of the crutch bounding object

A cylinder bounding object could be used, but following discussion with the developers of Webots™, due to possible internal problems, it was safer to use a box object.

V. VALIDATION OF BODY STATICS

The 3D modeling and the setup of the physical properties is done. The next step is to make sure the model answer the initial question of matching the real robot behaviors. In this part, a method is proposed to check the accuracy of the model static characteristics.

A. Experimental protocol

The main concept behind the validation of body statics is to avoid the dynamics to be involved in the process. To achieve this purpose, the position of the different servos has to be updated step by step to prevent any inertia factor to interfere the experiment.

The real quadruped robot is disposed in a stable position, then updating step by step the position of one servo, we check if the robot is falling down or not. The measured value is so called

the equilibrium value. Then reproducing the same experiment on the simulator, it's possible to compute the difference between the equilibrium value resulting from the real and the modeled experiment. A series of nine experiment is realized.

The Table 4 shows for each experiment the initial value of all the sixteen servos of the robot.

	Exp 1	Exp 2	Exp 3	Exp 4	Exp 5	Exp 6	Exp 7	Exp 8	Exp 9
PELVIS	513	513	205	819	513	513	513	513	513
F_L_1	513	513	0	0	513	513	513	513	513
F_R_1	513	513	513	513	513	513	513	513	513
F_L_2	205	819	819	819	867	513	513	513	513
F_R_2	819	205	513	513	513	166	513	513	513
F_L_3	513	513	513	513	513	513	513	513	513
F_R_3	513	513	513	513	513	513	513	513	513
B_L_1	513	513	513	513	513	513	513	513	513
B_R_1	513	513	513	513	513	513	513	513	513
B_L_2	580	444	513	513	166	513	513	513	513
B_R_2	444	580	819	819	513	867	513	513	513
B_L_3	513	513	513	513	513	513	513	513	513
B_R_3	513	513	513	513	513	513	513	513	513
N_1	513	513	513	513	513	513	513	513	513
N_2	513	513	513	513	513	513	513	819	205
HEAD	513	513	513	513	513	513	513	513	513

Table 4: Initial servo position for each of the static experiments. The gray cell shows the servos being updated during the experiment.

The results of the experiments done on the real robot have to be compared to the one done on the modeled one.

B. Results

The Table 5 summarizes the outcome of the 9 tests. The cross on the red cells states that the robot fall on the ground during the setup of the initial posture whereas the circle in the green cells depict that he robot didn't fall whatever was the position of the updated servo.

	Exp 1	Exp 2	Exp 3	Exp 4	Exp 5	Exp 6	Exp 7	Exp 8	Exp 9
<i>Real</i>	571	482	479	535	261	274	478	537	542
<i>Webots</i>	531	X	O	X	363	375	466	527	531
<i>Diff.</i>	40	X	O	X	102	101	12	10	11
<i>Diff.(deg)</i>	11.72	X	O	X	29.88	29.59	3.52	2.93	3.22

Table 5: Equilibrium value of the updated servo for each of the experiments

The issue of the experiment 1, 7, 8 and 9 can be accepted since the difference between the result obtained with the model and the one obtained within the simulator is below 15 degrees. Nevertheless, the issue of the of experiment 2, 3 and 4 need to be observed carefully (see Figures 19, 20 and 21).

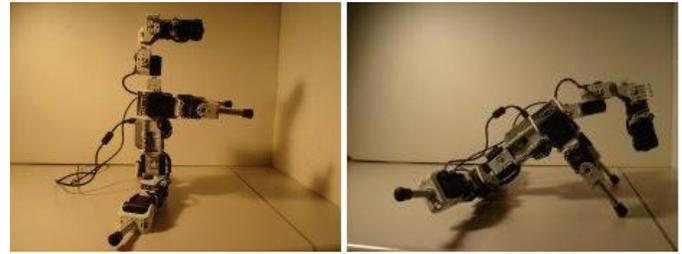


Illustration 18: Static test - Experiment 1



Figure 19: Static test - Experiment 2

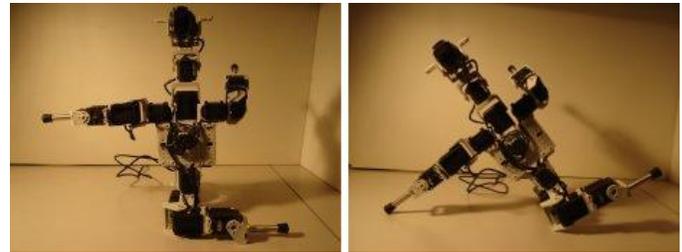


Figure 20: Static test - Experiment 3

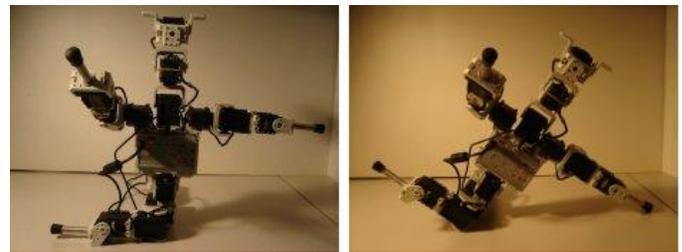


Figure 21: Static test - Experiment 4

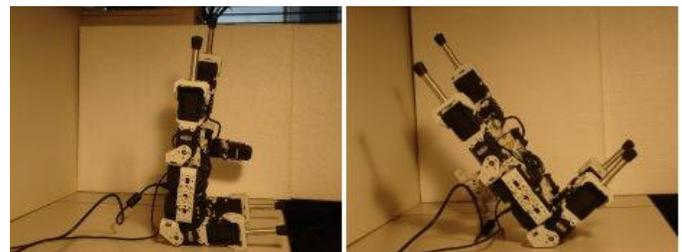


Figure 22: Static test - Experiment 5

In the context of experiment 3, the robot is “really” stable since in any position of the head the robot doesn't fall. Plus the

fact in experiment 2 and 3, it's not possible to set the robot in the initial posture. It's safe to tell that the center of mass impacting the most is the one of the CM5 unit since it is the heaviest element. The robot being really stable in the experiment 2, it's safe to assume the CoM is too much toward the left and/or bottom of the CM5 unit.

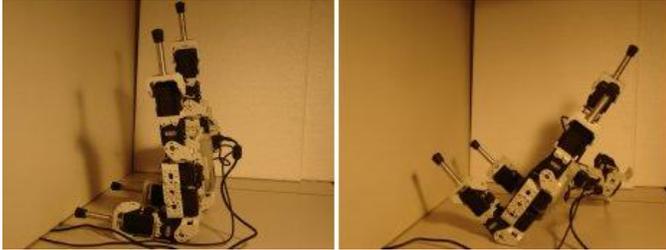


Figure 23: Static test - Experiment 6

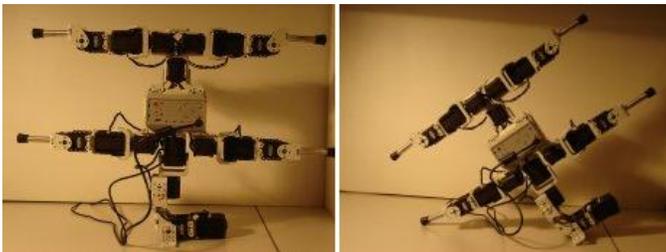


Figure 24: Static test - Experiment 8

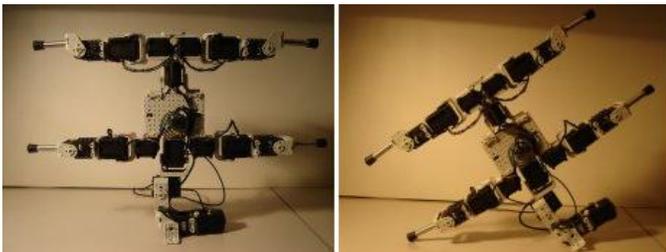


Figure 25: Static test - Experiment 9

To improve the issue of the experiments, the update of the position of the 16 center of mass can be a tedious and long process. The use of using a particle swarm optimization (see VI.B.2) is to optimize the location of the different center of mass could be considered. The function to maximize would be the following:

$$\phi = \frac{N}{\sum_{i=0}^N |real_i - webots_i|} \quad (2)$$

Where ϕ is the value to maximize, N the number of experiments, $real_i$ the outcome of the experiment with the real robot (it's a fixed parameter during the optimization), $webots_i$ the outcome of the experiment within the simulator.

VI. EXPERIMENTATION OF THE BODY DYNAMICS

A. Using inverse kinematics Model

1) Theoretical background

There are two ways of solving the inverse kinematics problems, either algebraically or geometrically. Only the solution will be given here. Further details are available in [29].

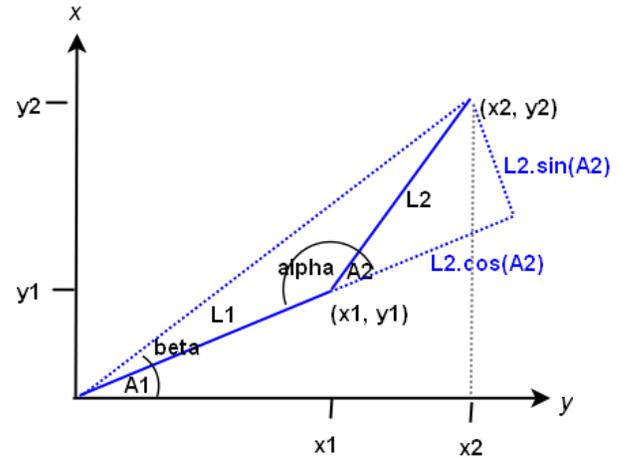


Figure 26: Determining angles A_1 and A_2 via geometric approach [29]

From Figure 26, it's possible to extract the value for A_1 and A_2 .

$$A_2 = \cos^{-1} \left(\frac{x^2 + y^2 - L_1^2 - L_2^2}{2 L_1 L_2} \right) \quad (3)$$

$$A_1 = \cos^{-1} \left(\frac{x^2}{\sqrt{x_2^2 + y_2^2}} \right) - \beta = \sin^{-1} \left(\frac{y^2}{\sqrt{x_2^2 + y_2^2}} \right) \quad (4)$$

where L_1 and L_2 are the respective length of both part of the robot leg. The extremity (x_2, y_2) being the bottom of the leg (foot), the point (x_1, y_1) being the axis of rotation of the lower leg (knee), and the origin of the coordinate system being the axis of rotation of the upper leg (shoulder).

$$x = L_1 \cos(A_1) + L_2 \cos(A_1 + A_2) \quad (5)$$

$$y = L_1 \sin(A_1) + L_2 \sin(A_1 + A_2) \quad (6)$$

Using (3) and (4), it's possible to compute A_1 and A_2 given a position of the foot.

2) Applied to the quadruped robot

The strength of the inverse kinematics model is to give the possibility of computing the different joint angles of a member knowing only the final trajectory of this one. Using an

elliptic trajectory for the leg extremity is known to be efficient for robotic walking. The parametric equations for an ellipse are (7) and (8).

$$x = h + a \cos(\phi) \quad (7)$$

$$y = k + b \sin(\phi) \quad (8)$$

Where ϕ is the phase between 0 and 2π , (h, k) determining the center of the ellipse, a and b being respectively the size of the radius for the x-axis and y-axis. The value of these parameters is relative to the anatomy of the real robot. Initially, as stated in [29], the rules of thumb to determine the equation parameters are:

- center of ellipse: $(h, k) = (0, -(L_1 + (L_2/2)))$
- radius on y-axis: $b = L_2/2$
- radius on x-axis: $a = 0.95(L_1/2)$

These rules set the maximum sized ellipse possible given the robot anatomy, not especially the most efficient.

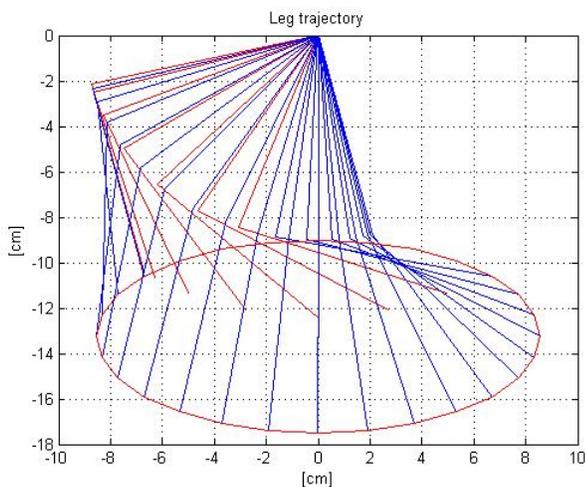


Figure 27: Leg trajectory over 28 steps

Applying the rules given in [29] the actual anatomy of the robot is not considered. This one isn't able to bend the lower leg over 90 degrees. The Figure 27 displays a red ellipse showing the lower leg extremity without considering the actual anatomy. The segment depicted in blue corresponds to the upper and lower leg having a possible position whereas the red segment corresponds to impossible position since the angle between the two segment is smaller than 90 degrees. The Figure 28(b) plots the servo values required to moved the leg given the current trajectory. A part of the red curve, as expected, is flat. This correspond to the maximum value the knee servo can't exceed.

To overcome this issue, the solution is to consider the actual robot anatomy. One of the initial rule of thumb needs to be updated. Setting the x-axis radius to $b = -k - \sqrt{L_1^2 + L_2^2}$ fixes the problem.

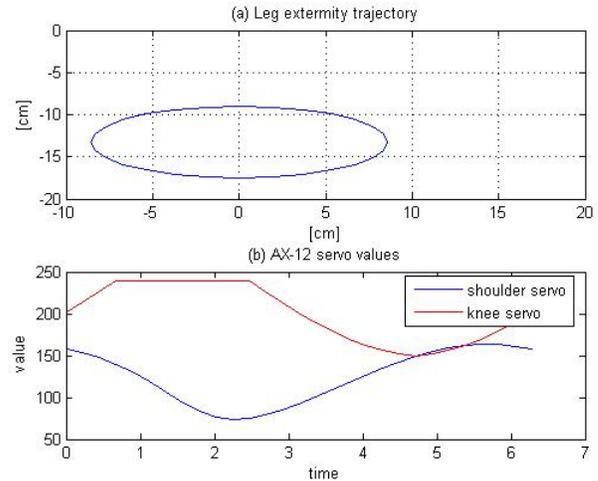


Figure 28: (a) Lower leg extremity trajectory, (b) AX12 values for shoulder and knee servo

3) Experimental framework

As explained in [30], the different possible gait in four-legged animal locomotion are depicted on Figure 29.

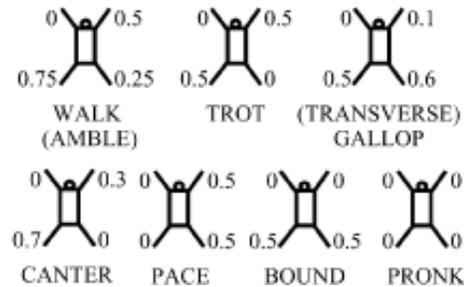


Figure 29: Gait used by four-legged walking animals, distinguished with respect to the relative relationships (in units of 2π) between their legs. The front left leg serves as reference. [30]

Using the inverse kinematics model described before, all the four-legged gaits listed above are tried at different frequency ranging from 0.5Hz to 2Hz.

4) Results

Looking at the Figure 30, the trot gait presents, from far and above all, the best performance. The performance is measured as a weighted sum of the absolute distance and the integrated distance done in between the 32 seconds of the simulation. A detailed description of the performance calculation is available in the part VI.B.3 page 16.

Freq [Hz]	Trot	Walk	Gallop	Canter	Pace	Bound	Prnk
0.5	8.68	2.63	1.25	1.11	0.93	1.19	2.98
1	11.29	3.16	0.33	0.27	2.88	0.39	0.68
1.5	11.06	4.17	0.84	0.83	2.81	0.58	2.43
2	12.41	3.80	6.32	1.70	3.55	0.26	1.41

 Move forward	 Walk backward
 Fall on the head or the side	 Stay on the spot

Figure 30: Performance as function of gait type and frequency

The Figures 31 to 34 show the trotting gait for the four frequency 0.5Hz , 1.0Hz , 1.5Hz and 2.0Hz . Since all the thumbnails are exactly generated every 62ms, the difference of performance is also visible. Indeed, taking all the bottom-right thumbnails of each figure and ordering them according to the position of the robot gives back the ranking of Figure 30.

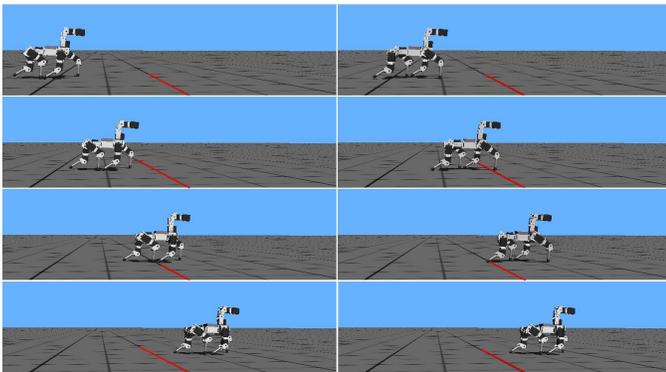


Figure 31: Trotting gait with $f = 0.5\text{Hz}$

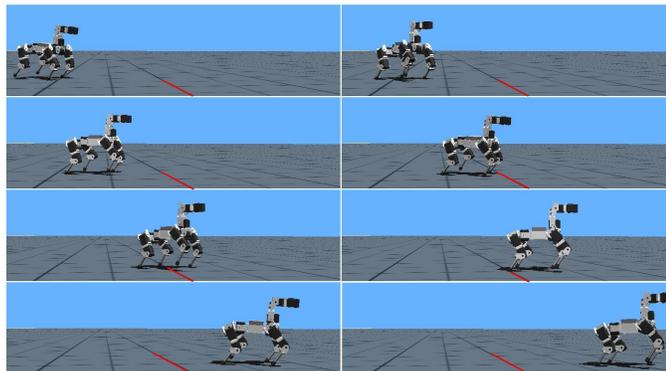


Figure 32: Trotting gait with $f = 1\text{Hz}$

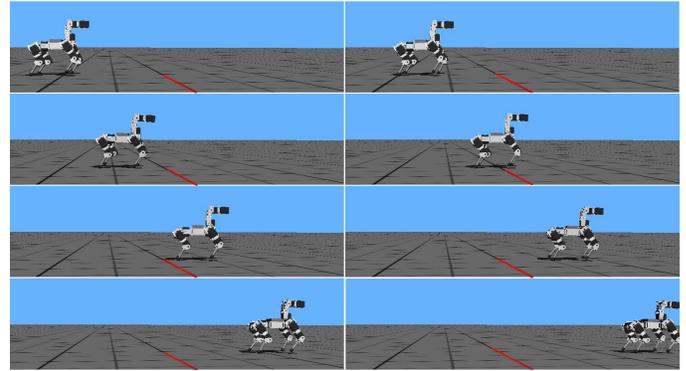


Figure 33: Trotting gait with $f = 2\text{Hz}$

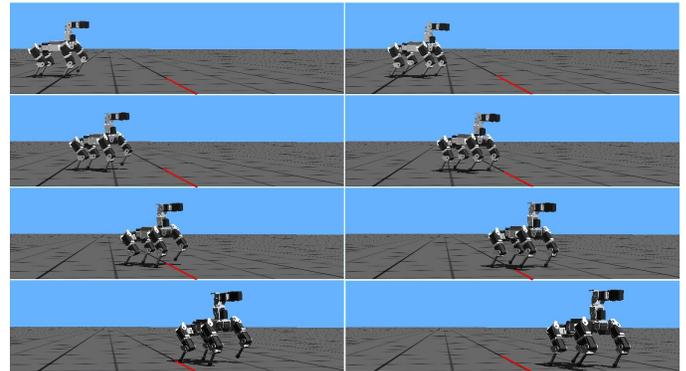


Figure 34: Trotting gait with $f = 1.5\text{Hz}$

B. Using CPG model optimized with PSO

The work of Yvan Bourquin[15] served as a basis for the design of the CPG models and their optimization. The subsections 1 and 2 are largely inspired from his work.

1) CPGs

a) Origins

“The concept of using non-linear oscillators to control robotic locomotion is inspired from biology. Experiments [13] showed that, in a decerebrated cat, the electrical stimulation of the brainstem is able to induce walking. Furthermore, an increase of the signal strength changes the walk velocity and the transition from a walking to a trotting gait happens autonomously. These experiments demonstrated that the brain is not involved in the generation of the rhythmic signals that produce locomotion in the cat.

Grillner[14] explained that the locomotory signals that produce sequences of muscle activation, such as walk, trot or gallop are generated by Central Pattern Generators (CPG) located in the spinal cord. These CPGs are neural circuits that generate oscillatory output from a tonic input coming from the brain. The brain appears to play a higher-level role such as regulating the initiation, velocity and termination of the locomotory activity.”[15]

b) Non-linear oscillator

“In animal locomotion, the oscillations of the joint angles produced by the muscular activity can have different waveforms. These waveforms are usually smooth: brutal transitions are uncommon. In order to facilitate numerical simulations, a strong simplification is to model locomotion as sinusoidal variations of the robot’s joint angles. In practice, sinusoidal signals are not flexible enough, because they do not allow a soft transition from one gait to another. For example, if a walk gait in a robot is controlled by sinusoidal motor signals, the transition to a different gait, say trot, requires the activation of different oscillations’ phases, amplitudes and frequencies. However, with sinusoidal signals, the transition from one gait to another is brutal and therefore cannot be carried out satisfactorily by the robot’s motors. Consequently an uncontrolled transition appears, during which the robot performs an undesired brutal movement and is subject to fall. This is in contrast to gait transitions in nature, which always occur smoothly. Furthermore, with sinusoidal signals, there is no simple way to incorporate sensory feedback.

To overcome this problem, non-linear oscillators were introduced as mathematical models of the natural CPGs [18]. The state of oscillators changes smoothly and therefore, gait transitions are soft. In addition, with oscillators, feedback can be incorporated in the simulation. For example, sensors can detect that a foot is in contact with the ground and a feedback signal can be injected into the oscillators.

The oscillator proposed in [18] is based on these differential equations:

$$\tau \dot{v} = -\alpha \frac{x^2 + v^2 - E}{E} v - x \quad (9)$$

$$\tau \dot{x} = v \quad (10)$$

where v and x represent the current state of the oscillator, E is a positive constant that represents the energy of the oscillator, α determines the rate of convergence towards the limit cycle and τ is the time constant that determines the oscillation’s frequency. This type of oscillator converges to a sinusoidal signal with amplitude \sqrt{E} and period $2\pi\tau$ [18]:

$$\tilde{x}(t) = \sqrt{E} \sin(t/\tau + \phi) \quad (11)$$

where ϕ depends on the initial conditions. This behavior is illustrated by the limit cycle presented on Figure 35 showing each run converges to the circular attractor of diameter $2\sqrt{E}=2$.”[15]

c) Synchronization

“By choosing the E and τ parameters, it is possible to control the amplitude and frequency of the oscillations. However, locomotion is efficient only when the phase shifts between the oscillations stays constant through time and

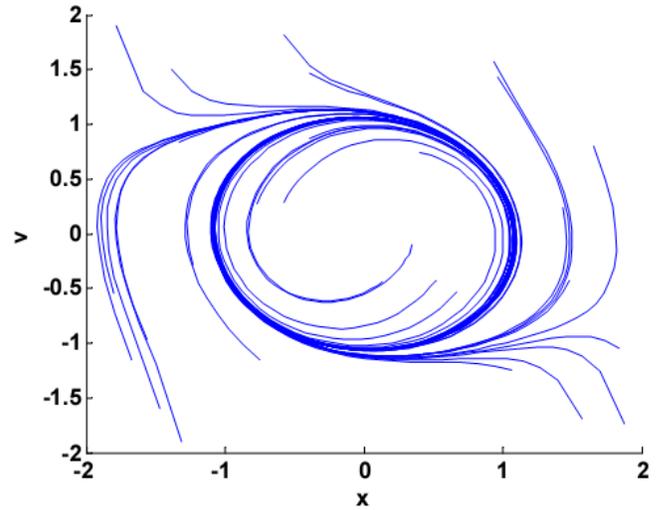


Figure 35: Limit cycles of a standalone oscillator obtained with 30 oscillations started with random initial conditions x and v in the range $[-2, 2]$. With $\alpha=0.7$ and $E=1$. [15]

therefore, a strict synchronization is required. In the model proposed in [18], synchronization is obtained by coupling the oscillators; a signal proportional to the sum of the state of every other oscillator is added into each oscillator. Equation (9) seen earlier, is now completed into equation (12) [18]:

$$\tau \dot{v}_i = -\alpha \frac{x_i^2 + v_i^2 - E}{E} v_i - x_i + \sum_j^N (a_{ij} x_j + b_{ij} v_j) \quad (12)$$

$$\tau \dot{x}_i = v_i \quad (13)$$

where a_{ij} and b_{ij} represents the strength of the coupling of the x and v states of oscillator j into the oscillator i .

Synchronization happens only when the uncoupled frequencies match approximately [19]. Figure 36 illustrates this fact: the frequency difference Δf of two uncoupled oscillators is plotted versus frequency detuning ΔF after coupling. If the uncoupled frequencies are too different, synchronization does not occur.

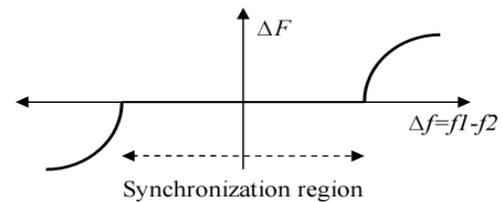


Figure 36: Frequency vs. detuning graph[19]

In order to facilitate synchronization the same time constant t is used for all the oscillators and therefore the uncoupled frequencies are similar. A frequency of 1Hz ($\tau=1/(2\pi)$) is chosen as baseline for all simulations. This is because it corresponds to the pace of an ordinary animal and it is slow enough for the physical robot’s servomotors to go once 180° back and forth. A stabilization period is necessary before the

frequencies become locked. The duration of the stabilization period depends on the coupling strength.

Figure 38 shows an example of synchronized oscillations. This stabilization period is bad because it results in disorganized steps of the robot. Shorter stabilization periods are wished and can be obtained by increasing the coupling strength. However, unlike standalone oscillators, the signals produced by coupled oscillators are not exact sine waves. Discrepancy with the sine increases with the coupling strength and furthermore, when the coupling becomes too strong the signals turn out to be chaotic (Figure 37) and unsuitable for controlling locomotion. For that reason, coupling strengths are suitable only within an appropriate range that must be determined.

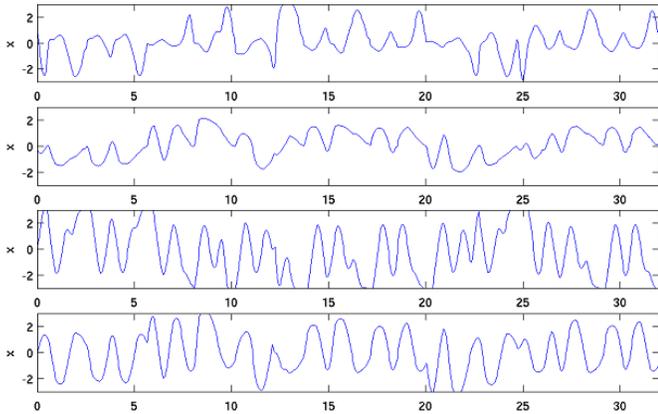


Figure 37: Chaotic oscillations resulting of too strong coupling. Plot of the x states of four coupled oscillators over a period of 32 seconds.[15]

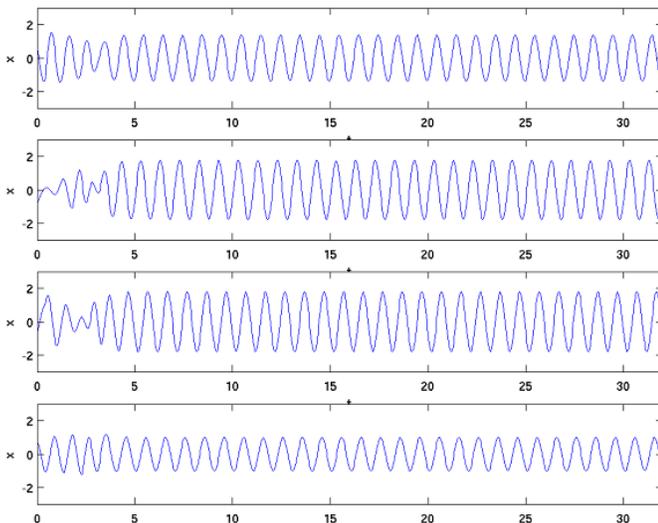


Figure 38: Example of synchronized oscillations Plot of the x states of four coupled oscillators over a period of 32 seconds: the initial stabilization period is visible. [15]

When multiple coupled oscillators are used, the resulting phase shifts between the oscillations is a function of the coupling strengths a_{ij} and b_{ij} , however several different

combination of a_{ij} and b_{ij} can produce the same phase shift. In fact, the exact outcome of a particular coupling combination cannot be predicted by any general theory [19]. Consequently, the coupling strengths must be optimized by the search algorithms.

A slight variation (14) of the original oscillators' model seen above was proposed in [20]. With this modification, the inputs are normalized and therefore, the "strength of the signal carried by a particular connection does not depend on the energy of the emitting oscillators" [Mojon 2004]. This modified model (14) is used in this project.

$$\tau \dot{v}_i = -\alpha \frac{x_i^2 + v_i^2 - E}{E} v - x + \sum_j \frac{(a_{ij} x_j + b_{ij} v_j)}{x_j^2 + v_j^2} \quad (14)$$

According to the simulation done, the coupling strengths were determined to be satisfactory in the range $[-0.7, -0.7]$. [15]

2) Particle Swarm Optimization

"Particle Swarm Optimization (PSO) was originally developed in 1995 by James Kennedy and Russell Eberhart [16]. Like genetic algorithms, PSO is based on a population that slowly converges towards one or more solutions. However, with PSO, the particles are preserved throughout the entire process; they do not die. Contrary to GA, which is based on competition for better chances of survival and reproduction, PSO uses a kind of cooperation between the particles. This is achieved through the exchange of the coordinates of the best solutions that have been encountered so far.

PSO's particles are simple search agents that "fly" through the search space. Whilst moving, they record the best position that they have discovered so far. They communicate with their neighbors and learn, from them, the best local solution. PSO is based on the concepts of social interaction or more exactly, the tendency of an individual to go his own way, as opposed to his tendency to follow his group's way. At every time step, a particle's flight direction is driven by three factors: first, its own inertial speed, second, its tendency to return to the best solution it has discovered so far, and third, the tendency to go towards the best solution discovered by its neighbors. This can be summarized in equation (15) which calculates the new flight speed of a particle at time $t+1$, and in equation (16) which calculates the new position a particle.

$$v_{id}(t+1) = \omega v_{id}(t) + \phi_1 \text{rand}() (p_{id} - x_{id}(t)) + \phi_2 \text{rand}() (p_{gd} - x_{id}(t)) \quad (15)$$

$$x_{id}(t+1) = x_{id}(t) + v_{id}(t) \quad (16)$$

where v is the speed of a particle, i is a particle index, d represents the d^{th} dimension in the parameter space, t is the discrete time index, ω is the particle speed inertia factor and where the function $\text{rand}()$ returns a uniformly distributed random number in the range $[0, 1]$. The coefficients ϕ^1 and ϕ^2 control the individual and social levels of confidence, e.g., how much a particle should follow its own best solution or his

group's best solution. Finally p_i is the best previous position of particle i , and p_g is the best previous position in the neighborhood of particle i . These principles are illustrated in Figure 39.

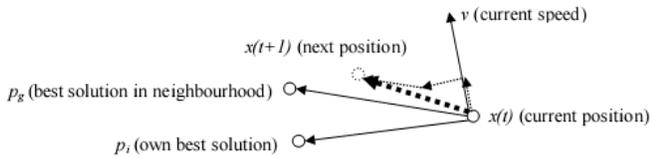


Figure 39: Principle of Particle Swarm Optimization

In PSO, we speak about the particles' neighborhood. A particle's neighborhood defines from which other particles the information will be received. The neighborhood size can vary from a few particles to the entire swarm. The neighborhood type does also vary: some PSO techniques are based on so-called social neighborhood, while others use geometrical neighborhood. In social neighborhood, the particles are associated with other particles from the beginning and their relationship is maintained throughout the process. A geometrical neighborhood is defined in accordance with the current particles "proximity" in the parameter space. In this case, the particle-to-particle distances needs to be recomputed at every iteration. The usually mentioned advantage of social neighborhood is its lower computational burden compared to geometrical neighborhood. However, in our case, geometrical neighborhood was preferred, because the processing power required by the optimization algorithm are insignificant anyway compared to that of the physics simulation.

The speed inertia factor ω can either be fixed or decreased during the optimization process. Some authors [17] suggest that a decreasing inertia factor gives better results, and so this approach is used here."[15]

3) Experimental framework

To summarize, in order to obtain suitable oscillations, different combinations of the τ , α , E , a_{ij} and b_{ij} , of equation (14) must be tried out. One option is to tune all parameters parameters with the optimization algorithms. However, in order to increase the likelihood of convergence it is better to hold down the dimensionality of the search space. Therefore, it is favorable to fix some parameters whilst the most relevant ones are kept free.

As explained in the previous paragraph, the coupling strengths a_{ij} and b_{ij} must be free because the oscillators' synchronization phase depends on them. The oscillation amplitudes controlled by E must also be free because it is not known beforehand how large joint movements should be.

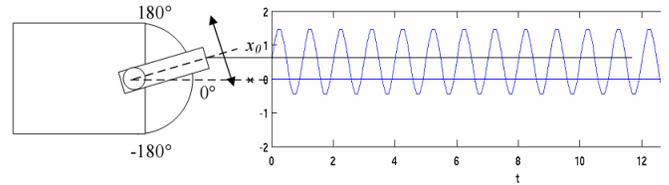


Figure 40: Module's oscillations around x_0 . [15]

Oscillations in the region of the module's 0° -angle (the horizontal position in Figure 40) are too restricted. For example in quadruped robots, it is known[15] that the "knee" joints should oscillate in a region below the 0° -angle. In general, this angle is not known beforehand and should therefore be another free parameter, that we call here x_0 .

Different oscillator coupling models were validated. The PSO used to optimize these models was initialized with a number of 50 particles. According to [15], this value gives interesting results. There is no defined method to determine the number of particle required to solve a given optimization problem, but based on previous experiments, the scientist can expect a convergence of the problem with a swarm size between 10 and 50 elements.

Computationally wise, a PSO working with 50 particles is high demanding. In fact, using one computer and thinking to optimize a simulation of 32 seconds. Considering the simulation runs in real time, the total time required should be around 107hrs. In case the simulation is slow down due to, for example, a high number of collision detection, this amount of time will be increased. In the case of our experiment, the Webots™ simulator was running in batch mode. Run a world in "batch mode" means there is no visual output of the simulation and the speed of the simulation is not constrained to real time. If the simulation can run faster, the simulator will do it. On the computer used for the simulation, Webots™ was able to run it 1.5 faster.

To speed up the optimization process, the PSO has been distributed over a network of 50 nodes. This way, doing 240 iterations of the PSO, the total time required was around 2 hours ($32 / 1.5 * 240$).

The implementation of the CPG realized by [15] was the starting point. That implementation didn't include the possibility to add bidirectional connection in the coupling of oscillators. Moreover, it was not possible to optimize only a subset of the servo modules. For example, sometimes it's interesting to optimize symmetrically the gait. The number of parameters will be decreased and the likelihood of the gait optimized versus a gait existing in nature will be increased.

To overcome these limitations, the initial implementation has been improved. In the configuration file relative to a controller, there is the description of all the servo involved in the optimization and there is the list of all the connection between these ones. A flag named "bidir" allows to enable or disable the mirroring of the coupling weight. In fact, a bidirectional connection is a connection were the weight of each direction are respectively opposed. To be able to optimize only a subset of the parameters relative to all the oscillator, a mapping mechanisms is developed. The output of

the PSO optimized gives a number X of values between 0 and 1. Then, inside the implementation of the controller, the parameters are mapped to their respective oscillators.

Bidirectional coupling are interesting if ones wants to increase the convergence speed of an oscillator and/or consider feedback from sensors.

In between all the experiments, the performance is measured the same way. As the crux of PSO is the optimization of the performance returned by a function. The function defined in [15] is used. Every evaluation lasts 32 seconds, in that time the robot has to go further possible. “However the performance could not be simply measured as the straight distance between the start and end location, because in some cases the robot makes a circle and stops close to where it started. In such cases, the performance evaluates poorly even though, only a tiny parameter change would be required in order to correct the robot’s bent trajectory. To overcome this problem, the cumulated or integrated ground distance was also integrated into the performance evaluation. Robot moving in a straight line should still be favored over zigzagging ones. Therefore, the performance needs to reflect both straight and the integrated distances. For this reason, the performance was calculated as the weighted sum of both, using the formula (17).”[15]

$$\phi = \alpha |\vec{p}_N - \vec{p}_1| + \beta \sum_{i=1}^{N-1} |p_{i+1} - p_i| \quad (17)$$

where ϕ represents the measured performance, where p_i is the i^{th} point sampled on the robot trajectory, where N is the total number of sampled point, and where α and β are coefficients that allow balancing the respective weights of the absolute and integrated distances. In our simulations these coefficients were set to $\alpha = 1$ and $\beta = 1$. In order to avoid granting the robots performance scores for plain vibrations, the trajectory points p_i are sampled at 1.0 second intervals such that a robot is always approximately in the same posture when sampling occurs.

Multiple optimization of CPGs are tried with a variable number of parameters, below are described and motivated the different choices of coupling models, their schematic diagrams are numbered from one to five.

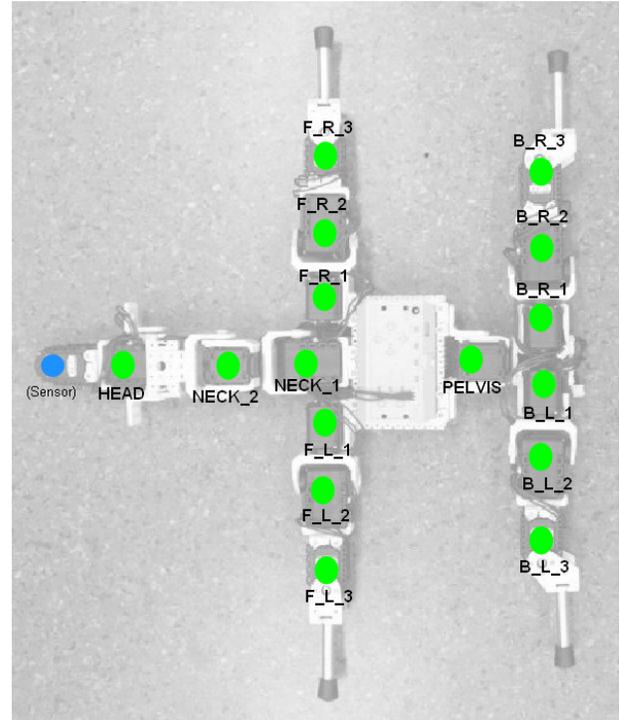


Figure 41: Name of the servos

The Figure 41 states the name of the different servos used through the experiments, descriptions and later on in the discussion of the results.

a) Experiment 1

All servos are linked with an oscillator, the coupling between them is unidirectional. The intrinsic frequency of all oscillator is the same. Each oscillator having 4 parameters (offset x_0 , amplitude \sqrt{E} , coupling weight w_a and w_b). The number of parameters to optimize is 64. CPG Model 1 (Figure 42) is used.

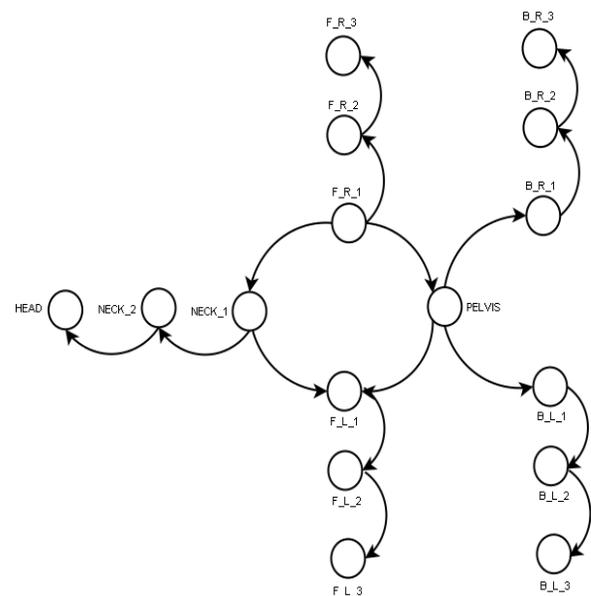


Figure 42: CPG Model 1

b) Experiment 2

The CPG Model 2 is used (see Figure 43). We can consider the movement of the HEAD and NECK_2 servos as useful only in term of vision stabilization capabilities. In the framework of this optimization problem, one's could argue that only the impact of the weight of the head on the dynamics of the body is important, the servo HEAD and NECK_2 being used only to keep the vision stable or to look on the right or left, they are kept in position. Only the NECK_1 servo is used to move the head. The connection within the body are all bidirectional, that way the oscillator of the torso and pelvis are supposed to converge faster. This experiment is improved compared to the first one. Observing how a dog walks or runs, we can tell that the joints of the leg move the same but are only shift in time. Assuming this hypothesis is correct, the size of search space is decreased and the chance to converge to a valuable optimum performance maximized. Applying that concept, the set of parameters of the following pairs of oscillators are considered identical (F_{L_3}, F_{R_3}), (F_{L_2}, F_{R_2}), (F_{L_1}, F_{R_1}), (B_{L_3}, B_{R_3}), (B_{L_2}, B_{R_2}), (B_{L_1}, B_{R_1}). It also safe to tell, while a dog is walking, that the frequency of the head and pelvis could be different from the the other joints. That's why this time, the frequency of the PELVIS and NECK_1 oscillator has been optimized between a value of 1Hz and 2Hz. The number of parameters to optimized is now 33.

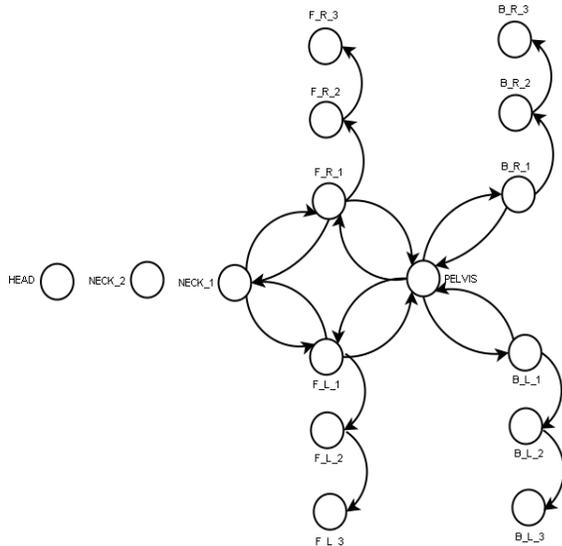


Figure 43: CPG Model 2

c) Experiment 3

This experiment is similar to the previous one except one detail concerning the frequency of the PELVIS and NECK_1 oscillator. The frequency parameter is now removed, the PELVIS and HEAD will oscillate at the same frequency the rest of the oscillators do. The number of parameters is 32.

d) Experiment 4

Observing real dogs walking, it seems their pelvis is moving much more from left to right than up and down. In this experiment, the influence of the PELVIS servo has been annihilated. In the same time the NECK_1 has been removed since the pelvis and the head are considered to be linked in frequency. The CPG Model 3 (see Figure 44) being used, the total number of parameter to optimized falls to 24.

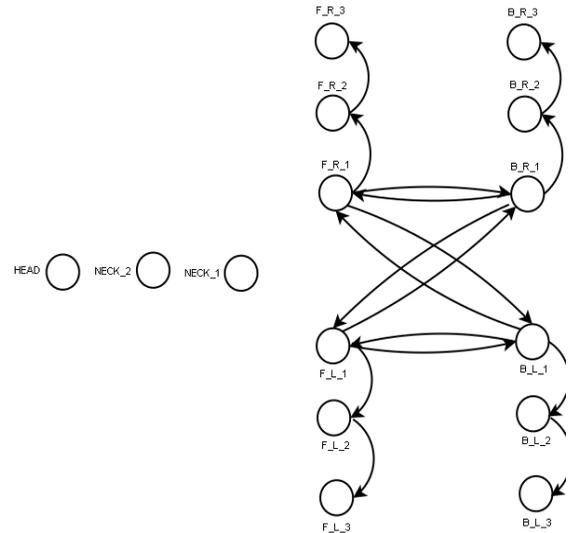


Figure 44: CPG Model 3

e) Experiment 5

As in experiment 4, the PELVIS and NECK_1 oscillator are taken away. CPG Model 4 (see Figure 45). An additional hypothesis is done, real dog can hardly move their front leg laterally far away from their torso. As simplification, in this experiment, the $F_{R_2}, F_{L_2}, B_{R_2}$ and B_{L_2} servos are kept in fixed position, keeping their value as when the dog stand up. Since the pair of oscillators (F_{L_1}, F_{R_1}) and (B_{R_1}, B_{L_1}) are identically optimized, the number of parameters is 16.

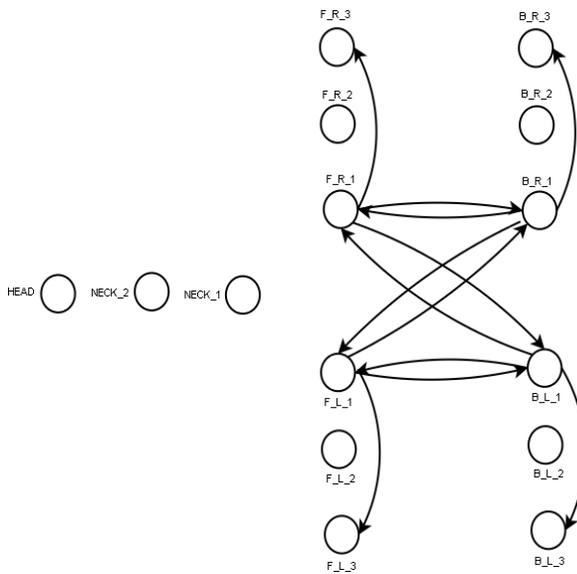


Figure 45: CPG Model 4

f) Experiment 6

Keeping the same hypothesis as in experiment 5, the pairs of oscillators (F_L_1, F_R_1) and (B_R_1, B_L_1) have now their own set of optimized parameters. The same model is used (see Figure 45). Since the front and back top leg oscillator are unpaired, the number of parameters is 24.

4) Results and discussion

The thumbnails showing the best performance obtained so far within the context of a given experiment are order from left to right and top to bottom. These one were generated over a sequence of approximately 5 seconds. The aim of these sequence is too visualize the way the robot moves.

As we can see in the plots depicted in the subsection specific to each experiments, a maximum of 240 iterations for the PSO seems to be a value allowing to converge toward a solution. Indeed, due to the random initialization of the 50 particles of the swarm, the first iterations give wrong performance and then, as the number of iterations increased, the performance increases until it reached an asymptotic limit. The performance for all experiments oscillated between approximately 7 and 17.

a) Experiment 1

Optimized a function having 64 parameters is quite a challenge, using a systematic search the cardinality of the space would have been 2^{64} which implies approximately $1.84 \cdot 10^{19}$ sets of possible parameters. A way to distribute such a simulation, doing a systematic search, could be to ask each inhabitants of the world[25] to run $2.78 \cdot 10^9$ evaluations. Then, considering each one of them has a computer able to run the simulation 32 times faster. The

amount of time required, if we distribute the systematic search to all world inhabitants, would be 88 years .

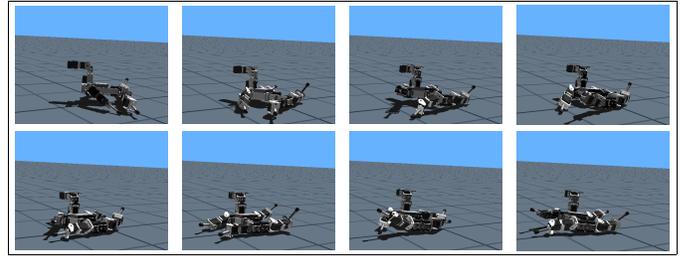


Table 6: Experiment 1 - Best performance

Observing Table 6, we see that the robot moves hardly forward, like it was swimming on the ground. The gait obtained is not similar to what a real dog does. Nevertheless, it is safe to assume the optimization could be more successful doing hypothesis on the coupling. That way, the cardinality of the problem could be decreased.

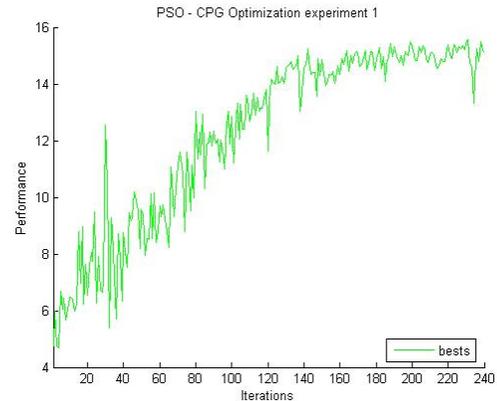


Figure 46: Experiment 1 - Plot of PSO performance over 240 iterations.

b) Experiment 2

Compared to the experiment 1, the cardinality of the problem is divided by two. Doing the same analogy about the number of world inhabitants, each one of them would have to run only one simulation for less than a seconds.

Observing Table 7, we can see the dog is “walking” backward. Since the control unit of the robot is heavy compared to the rest of the body, this one being located toward the torso, a backward gait could be a possible solution to overcome the stability problem. That hypothesis seems to be suitable to justify the behavior observed.

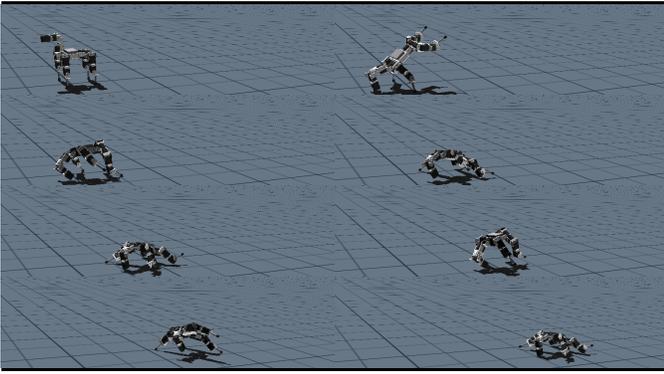


Table 7: Experiment 2 - Best performance

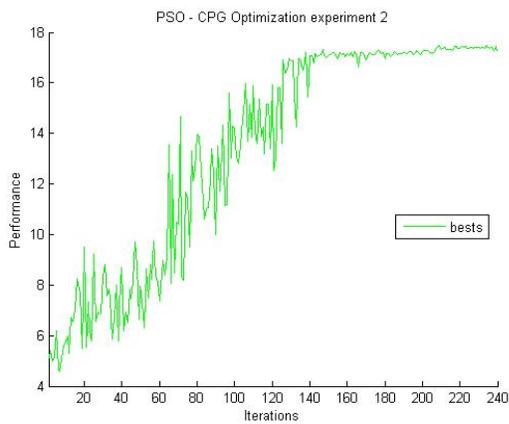


Figure 47: Experiment 2 - Plot of PSO performance over 240 iterations.

c) Experiment 3

Looking at Table 8, the dog is, in opposition to what was observed in the previous experiment “walking” forward. The fact the direction is different is not especially in contradiction with the result observed previously. Indeed, in the present case, the dog keeps the back leg lower while moving, this way the center of gravity is kept close to the ground and the possible problems of stability are avoided.

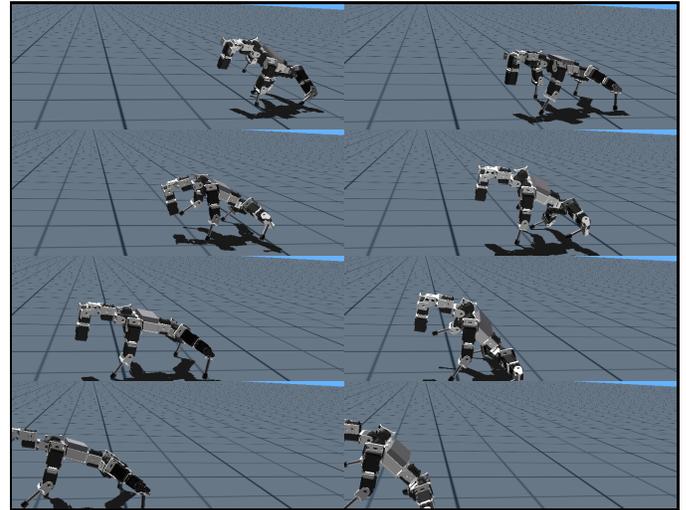


Table 8: Experiment 3 - Best performance

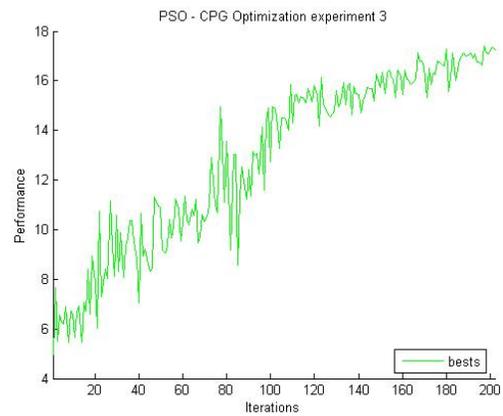


Figure 48: Experiment 3 - Plot of PSO performance over 202 iterations.

d) Experiment 4

As showed on Table 9, the robot moves backward and on the left keeping his back right leg up. The resulting gait is a bit chaotic. This leg being up, the dog as to keep his balance on the three other legs while moving. To overcome a possible fall on the side with one leg, the dog moves on the opposite side with a tendency to walk backward to compensate the position of center of gravity being near the torso.

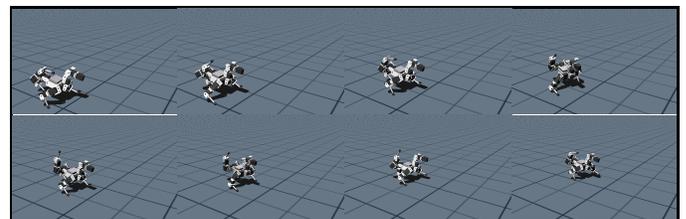


Table 9: Experiment 4 - Best performance

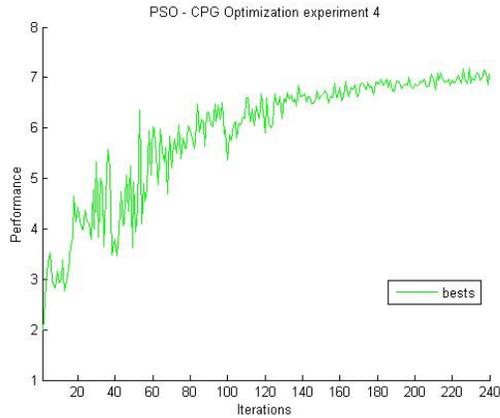


Figure 49: Experiment 4 - Plot of PSO performance over 240 iterations.

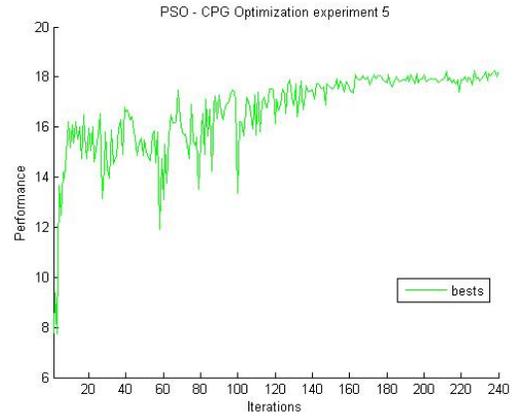


Figure 50: Experiment 5 - Plot of PSO performance over 240 iterations.

e) Experiment 5

In this experiment, the dog is moving forward. Front and back legs are respectively synchronized. Since the oscillator parameters of the front left leg are mirrored to the right leg and respectively the same for the back leg. As in a real dog gallop gait, the phase shift between back and front oscillator is the same. The center of gravity is closer to the torso, then when the dog is falling, his head and the fact the front leg are bended allows him to go back to his position. Indeed, while at this position, the movement of the back leg backward give enough inertia to the body to make it stand back on the four legs.

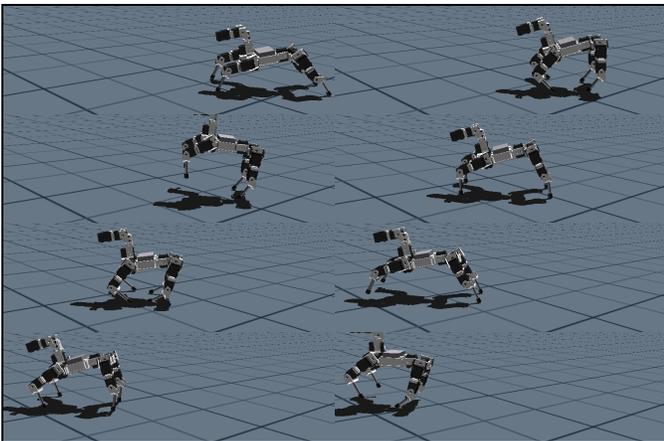


Table 10: Experiment 5 - Best performance

f) Experiment 6

Similarly to the experiment 5, the shoulder servos are in fixed position keeping the legs close to the body and avoiding gait where the dog is almost “swimming” on the ground. In opposition to the previous experiment, the F_R_1, F_L_1, B_L_1 and B_R_1 oscillators are optimized independently allowing to have different weight between them so different phases difference. As in a real dog walking gait, the four legs are phase shifted.

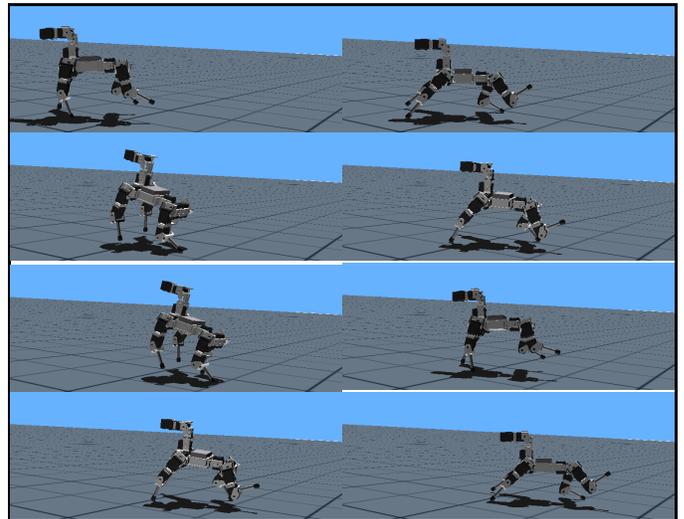


Table 11: Experiment 6 - Best performance

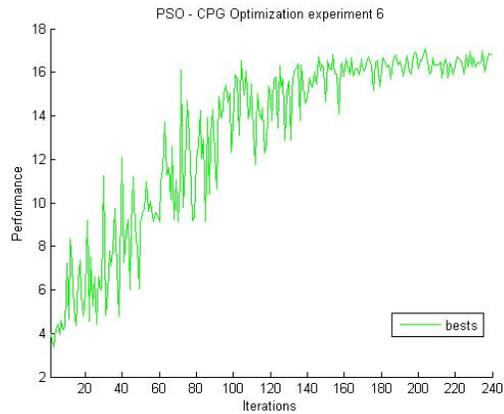


Figure 51: Experiment 6 - Plot of PSO performance over 240 iterations.

VII. CONCLUSION

It is believed that the model of the Bioloid robot has been successfully designed. The physics were applied to the model and the static validation of the model is on average good enough. Despite the efforts made, a lot of work remains outstanding and multiple aspects can be improved

Developing a Bioloid specific modeling language helping at describing easily the connection in between the existing pre-defined shape would be a cornerstone in the future development of model.

Validating the approximation of crutch bounding object is good enough and that the physic parameters concerning the coulomb friction and the bounciness are matching the behavior of the real quadruped robot. And later on, developing a test allowing to check if Webots™ behaves the same way when the simulation are done in “run”, “fast” and “batch” mode would consolidate the results obtained so far.

As discussed, with the Australian team responsible for the hardware development, the initial design of the robot has been updated to give the quadruped an “aibo” style of walking. Wedges were added between the knee and the lower leg to add a permanent offset of 18 degrees. Adapting the inverse kinematics model and comparing the performance of the gaits given this new anatomy of the quadruped could be interesting. Moreover, the CPG optimization could also be redone considering these wedges.

According to the outcome of the different optimization, following the parameters optimized and the coupling model of the CPG, it could be interesting to implement a more sophisticated central pattern generator more sophisticated. For example, the inverse kinematics model developed could be implemented with oscillators being coupled at the same time to the pelvis and neck. The function to measure the performance could also be improved. Favored forward gait, add bounding to the possible position of a body part. For example, give credit if the head stay above a given line or stay horizontal.

The most efficient gait implemented was discovered to be the trotting one obtained after using the inverse kinematics model. However, far from being perfect the robot do not go always straight. Given a target aim, the robot could adapt and try to add correction to his current direction to keep the orientation. The turning right and left operations could also be extrapolated from the improved gait.

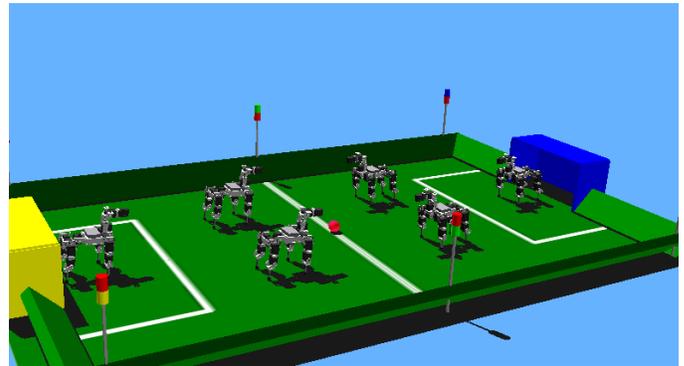


Figure 52: Six quadruped playing on a soccer field

As depicted on Figure 52, multiple robot are able to evolve simultaneously on the soccer field matching the demo requirements [31]. Additional work has to be conducted to allow the quadruped to detect the ball within his environment and try to kick it.

A plus to the current implementation could be the ability to cross-compile the Webots™ controller into CM5 controllers.

ACKNOWLEDGMENT

I gratefully acknowledge the technical support and the advices of Prof. Auke Jan Ijspeert, Olivier Michel, Yvan Bourquin, Peter Turner and Robin Fisher in the design and implementation of the quadruped robot model. I would like to acknowledge Laurent Lessieux for his work on the X file processing. Finally, I also acknowledge Matteo Thomas DeGiacomi and Alessandro Crespi for their advices and help in the everyday lab life. This work was made possible thanks to the Biologically Inspired Robotic Group.

REFERENCES

- [1] O. Michel, *Cyberbotics Ltd – Webots™: professional mobile robot simulation*. International Journal of Advanced Robotic Systems (2004) Volume 1 Number 1: pp. 39-42.
- [2] S. Russell, *Open Dynamic Engine (ODE): open source and high performance library for simulating rigid body dynamics*. - <http://www.ode.org/>
- [3] Research projects, game or various tools that are using ODE. - <http://www.ode.org/users.html>

- [4] Atmega128, 128-Kbyte self-programming Flash Program Memory, 4-kbyte SRAM, 4-kbyte EEPROM, 8 Channel 10-bit A/D-converter. JTAG interface for on-chip-debug. - <http://www.atmel.com>
- [5] Bioloid User Guide - *Understanding ID, Address and data* p27 - <http://www.tribotix.info/Downloads/Robotis/Bioloid/Bioloid%20User%27s%20Guide.pdf>
- [6] A extended Object file format description - http://www.eg-models.de/formats/Format_Obj.html
- [7] http://en.wikipedia.org/wiki/Wavefront_Technologies
- [8] Text File format of The Persistence of Vision Raytracer, or POV-Ray which is a ray tracing program available for a variety of computer platforms. - <http://en.wikipedia.org/wiki/POV-Ray>
- [9] Art of Illusion, free and open-source java-based modeling and rendering studio - <http://aoi.sourceforge.net/>
- [10] Virtual Reality Modeling Language - <http://en.wikipedia.org/wiki/VRML>
- [11] Webots™ Nodes Chart - <http://www.cyberbotics.com/cdrom/common/doc/webots/reference/section4.1.html>
- [12] Webots™ Reference Manual -Physics Node - <http://www.cyberbotics.com/cdrom/common/doc/webots/reference/section2.33.html>
- [13] M. L. Shik, F. V. Severin, G. N. Orlovskii, *Control of Walking and Running by Means of Electrical Stimulation of the Mid-Brain*. Biophysics, 11:756-765, 1966.
- [14] S. Grillner, *Neurobiological Bases of Rhythmic Motor Acts in Vertebrates*. Science, New Series, Vol. 228, No. 4696 (Apr. 12, 1985), 143-149.
- [15] Y. Bourquin, *Self-Organization of Locomotion in Modular Robots*. MSc Dissertation, p16, p26
- [16] J. Kennedy, R. Eberhart, *Particle Swarm Optimization*. Proceedings of the 1995 IEEE International Conference on Neural Networks, pp. 1942-1948, IEEE Press.
- [17] Y. Shi, and R. C. Eberhart, (1998). *Parameter selection in particle swarm optimization*. In Evolutionary Programming VII: Proc. EP98, New York: Springer-Verlag, pp. 591-600.
- [18] A. J. Ijspeert, J.-M. Cabelguen (2003), *Gait transition from swimming to walking: investigation of salamander locomotion control using non-linear oscillators*. In Proceedings of Adaptive Motion in Animals and Machines, 2003.
- [19] A. Pikovsky, M. Rosenblum, and J. Kurths, (2001). *Synchronization, a universal concept in nonlinear sciences*. Cambridge Nonlinear Sciences Series 12.
- [20] S. Mojon, (2004). *Using nonlinear oscillators to control the locomotion of a simulated biped robot*. Unpublished Diploma Thesis. <http://birg.epfl.ch/page44565.html>
- [21] *User manual of Dynamixel Sensor module AX-S1*, release of 06-14-2006
- [22] *User manual of Dynamixel module AX-12*, release of 06-14-2006
- [23] Microsoft Direct X API - <http://en.wikipedia.org/wiki/DirectX>
- [24] KDE development Environment - <http://www.kdevelop.org/>
- [25] Bioloid QuickStart “Comprehensive Kit” Manual - [http://www.tribotix.info/Downloads/Robotis/Bioloid/QuickStart\(ComprehensiveKit\).pdf](http://www.tribotix.info/Downloads/Robotis/Bioloid/QuickStart(ComprehensiveKit).pdf)
- [26] Rotation representation - [http://en.wikipedia.org/wiki/Rotation_representation_\(mathematics\)](http://en.wikipedia.org/wiki/Rotation_representation_(mathematics))
- [27] Webots™ Reference Manual - Servo Node - <http://www.cyberbotics.com/cdrom/common/doc/webots/reference/section2.37.html>
- [28] Number of world inhabitants – <http://www.worldpopclock.com>
- [29] P. Turner, *Mathematics required for Legged Robotic Motion*, rev 1, September 2006
- [30] R. M. Alexander, *Locomotion of Animals*. Glasgow, London, U.K.:Blackie, 1982.
- [31] Rules for the Four Legged Robot League - <http://www.tzi.de/4legged/pub/Website/Downloads/Rules2007.pdf>

APPENDIX

All the videos, sources code and data files acquired during the different experiments will be posted on the project website. The current project page is <http://wiki.epfl.ch/wsrl>. This page is susceptible to change in the future, in case the link is dead. You will be able to look for additional information using the Biologically Inspired Robotic Group (BIRG) page at <http://birg.epfl.ch> or <http://birg.epfl.ch/page32024.html>.

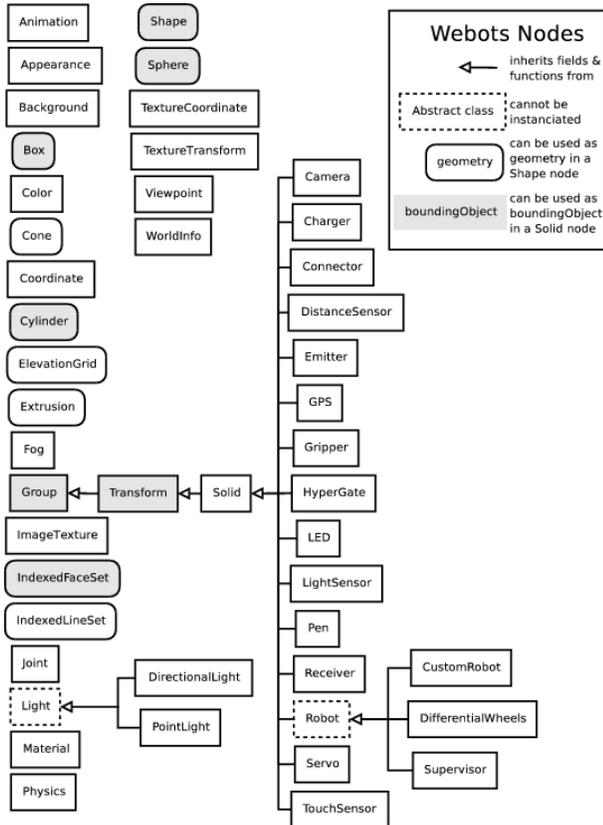


Figure 53: Webots™ Nodes Chart outlining all the nodes available to build Webots worlds [11]

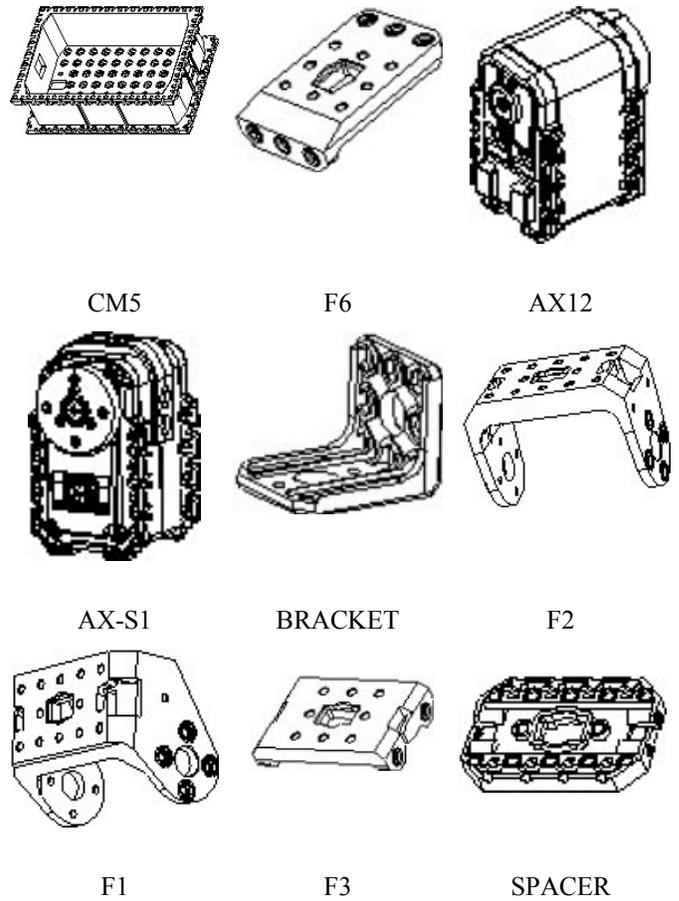


Table 13: List of shape needed to build the quadruped robot.

```
function [e1, e2, e3, theta] = euler2vrm1(alphaX, alphaY, alphaZ)
% see
http://en.wikipedia.org/wiki/Rotation\_representation\_%28mathematics%29

%convert degree to pi-rad
alphaX = alphaX * pi/180;
alphaY = alphaY * pi/180;
alphaZ = alphaZ * pi/180;

% compute the Direction Cosine Matrix (DCM) from euler angles
Ax = [1 0 0; 0 cos(alphaX) -sin(alphaX); 0 sin(alphaX) cos(alphaX)];
Ay = [cos(alphaY) 0 sin(alphaY); 0 1 0; -sin(alphaY) 0 cos(alphaY)];
Az = [cos(alphaZ) -sin(alphaZ) 0; sin(alphaZ) cos(alphaZ) 0; 0 0 1];
A = Az * Ay * Ax;

%check DCM back to euler
atan2(A(3,1), A(3,2)) * 180/pi
acos(A(3,3)) * 180/pi
-atan2(A(1,3), A(2,3)) * 180/pi

% compute the euler angle / rotation axis from the DCM
theta = acos( (A(1,1) + A(2,2) + A(3,3) - 1) / 2);
e1 = ( A(3,2) - A(2,3) ) / ( 2 * sin(theta) );
e2 = ( A(1,3) - A(3,1) ) / ( 2 * sin(theta) );
e3 = ( A(2,1) - A(1,2) ) / ( 2 * sin(theta) );
```

Table 12: Matlab script to convert from Euler angle notation to VRML angle(euler axis/angle) notation

```
function [alphaX, alphaY, alphaZ] = vrm12euler(X, Y, Z, theta)
% see
http://en.wikipedia.org/wiki/Rotation\_representation\_%28mathematics%29

% compute the Direction Cosine Matrix (DCM) from euler angle / rotation axis
E = [X Y Z]';

%normalized vector
E = E / norm(E);

A = eye(3) * cos(theta) + (1 - cos(theta)) * E*E' - [0 -Z Y; Z 0 -X; -Y X 0] * sin(theta);

%transpose matrix
A = A';

% compute the euler angle / rotation axis from the DCM
alphaX = acos(A(3,3));
alphaY = atan2(A(3,1), A(3,2));
alphaZ = pi - atan2(A(1,3), A(2,3));

%convert pi-rad to degree
alphaX = alphaX * 180/pi;
alphaY = alphaY * 180/pi;
alphaZ = alphaZ * 180/pi;
```

Table 14: Matlab script to convert from VRML angle(euler axis/angle) notation to Euler notation

```
ANOUKA Supervisor
CM5_TRANS Trans { CM5_SHAPE Shape }
CM5UNIT_TRANS Trans { CM5UNIT_SHAPE Shape }
F3_FRONT_LEFT_1_TRANS Trans
F3_SHAPE Shape
F3_FRONT_LEFT_2_TRANS Trans
```

```

AX12_FRONT_LEFT_1_TRANS Trans
AX12_SHAPE Shape
FRONT_LEFT_1_SERVO Servo
  F1_FRONT_LEFT_1_TRANS Trans { F1_SHAPE Shape }
FRONT_LEFT_2_SERVO Servo
  AX12_FRONT_LEFT_2_TRANS Trans { AX12_SHAPE }
  F3_FRONT_LEFT_3_TRANS Trans
  F3_CROSSED_GRP Group { F3_SHAPE }
  F3_CROSSED1_TRANS Trans { F3_SHAPE }
  AX12_FRONT_LEFT_3_TRANS Trans { AX12_SHAPE }
FRONT_LEFT_3_SERVO Servo
  F2_TRANS Trans { F2_SHAPE Shape }
  CRUTCH_FRONT_LEFT_TRANS
  CRUTCH_DISC_SHAPE
  CRUTCH_TRANS Trans { CRUTCH_SHAPE }
  CRUTCH_PLASTIC_TOP_TRANS Trans
{ CRUTCH_PLASTIC_TOP_SHAPE }
  CRUTCH_PLASTIC_BOTTOM_TRANS Trans
{ CRUTCH_PLASTIC_BOTTOM_SHAPE }
F3_NECK1_TRANS Trans
  F3_CROSSED_GRP
  AX12_NECK1_TRANS Trans { AX12_SHAPE }
NECK1_SERVO Servo
  F2_NECK1_TRANS Trans { F2_SHAPE }
NECK2_SERVO Servo
  AX12_NECK2_TRANS Trans { AX12_SHAPE }
  F6_NECK2_LEFT_TRANS Trans { F6_SHAPE }
  F6_NECK2_RIGHT_TRANS Trans { F6_SHAPE }
  F2_NECK2_CENTER_TRANS Trans
  F2_SHAPE
  BRACKET_NECK2_CENTER_TRANS Trans
  BRACKET_SHAPE Shape
  F1_NECK2_CENTER_TRANS Trans
  F1_SHAPE
  BRACKET_NECK2_LEFT_EAR_TRANS Trans
{ BRACKET_SHAPE }
  BRACKET_NECK2_RIGHT_EAR_TRANS Trans
{ BRACKET_SHAPE }
  HEAD_SERVO Servo
  AX12_HEAD_TRANS Trans { AX12_SHAPE }
  F3_HEAD_TRANS Trans
  F3_CROSSED_GRP
  AX12_HEAD_2_TRANS Trans { AX12_SHAPE }
  F3_FRONT_RIGHT_1_TRANS Trans { F3_FRONT_LEFT_1_TRANS }
  AX12_FRONT_RIGHT_1_TRANS Trans { AX12_SHAPE }
FRONT_RIGHT_1_SERVO Servo
  F1_FRONT_RIGHT_1_TRANS Trans { F1_SHAPE }
FRONT_RIGHT_2_SERVO Servo
  AX12_FRONT_RIGHT_2_TRANS Trans
  AX12_SHAPE
  F3_FRONT_RIGHT_3_TRANS Trans
  F3_CROSSED_GRP
  AX12_FRONT_RIGHT_3_TRANS Trans { AX12_SHAPE }
FRONT_RIGHT_3_SERVO Servo
  F2_TRANS Trans { F2_SHAPE }
  CRUTCH_FRONT_LEFT
F3_PELVIS_LEFT_1_TRANS Trans
  F3_SHAPE
  BRACKET_PELVIS_LEFT_1_TRANS Trans
  BRACKET_SHAPE
  F6_PELVIS_LEFT_1_TRANS Trans { F6_SHAPE }
  AX12_PELVIS_TRANS { AX12_SHAPE }
PELVIS_SERVO Trans
  PELVIS_1_TRANS Trans
  SPACER_SHAPE Shape
  F3_PELVIS_LEFT_1_TRANS Trans { F3_SHAPE }
  AX12_BACK_LEFT_1_TRANS Trans { AX12_SHAPE }
  F3_PELVIS_RIGHT_1_TRANS Trans { F3_SHAPE }
  AX12_BACK_RIGHT_1_TRANS Trans { AX12_SHAPE }

PELVIS_2_TRANS Trans
  SPACER_SHAPE
  F6_PELVIS_LEFT_1_TRANS Trans { F6_SHAPE }
  F6_PELVIS_RIGHT_1_TRANS Trans { F6_SHAPE }
BACK_LEFT_1_SERVO Servo
  F1_BACK_LEFT_1_TRANS Trans { F1_SHAPE }
BACK_LEFT_2_SERVO Servo
  AX12_BACK_LEFT_2_TRANS Trans { AX12_SHAPE }
  F3_BACK_LEFT_3_TRANS Trans
  F3_CROSSED_GRP
  AX12_BACK_LEFT_3_TRANS Trans { AX12_SHAPE }
BACK_LEFT_3_SERVO Servo
  F1_TRANS Trans { F1_SHAPE }
  CRUTCH_BACK_LEFT_TRANS
  CRUTCH_FRONT_LEFT
BACK_RIGHT_1_SERVO Servo
  F1_BACK_RIGHT_1_TRANS Trans { F1_SHAPE }
BACK_RIGHT_2_SERVO Servo
  AX12_BACK_RIGHT_2_TRANS Trans { AX12_SHAPE }
  F3_BACK_RIGHT_3_TRANS Trans

```

```

F3_CROSSED_GRP
AX12_BACK_RIGHT_3_TRANS Trans { AX12_SHAPE }
BACK_RIGHT_3_SERVO Servo
  F1_TRANS Trans { F1_SHAPE }
  CRUTCH_BACK_LEFT
F3_PELVIS_RIGHT_1_TRANS Trans
  F3_PELVIS_LEFT_1_TRANS

```

Table 15: Simplified representation of the robot model, the gray lines show the servo nodes.