ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

BIOLOGICALLY INSPIRED
ROBOTICS GROUP (BIRG)

Java Applet
for the Locomotion Controller
of the Salamander Robot

Semester Project – Winter 2005–2006

Sarah Marthe

Professor Auke J. Ijspeert
Supervisor: Alessandro Crespi

February 2006

Abstract

In the context of research on animal locomotion, a salamander robot and its locomotion controller are currently under development at the Biologically Inspired Robotics Group (BIRG) at the EPFL.
This document presents a Java applet for this controller, developed as a semester project in the purpose of making the functioning of the controller and the effects of some parameter visible and accessible for other researchers and scientists. This report is divided into seven parts. There is first an introduction to the aims of the project, followed by some background information about the controller itself and its situation in a more general research context. The third part details the design of the Java applet, carried from the C implementation of the controller and the fourth part presents the graphical user interface and the behaviours and results observable from it. The fifth and sixth parts explain the problems encountered during this project and detail some ideas for further work. Finally, the report is concluded with some personal assessment about the goals achieved and feelings about the whole project.

Table of Contents

I – Introduction 1
II – Background 2
II – 1 The real salamander 2
II – 2 The locomotion controller 3
II – 2.1 Nonlinear oscillator model 3
II – 2.2 Central pattern generator (CPG) 5
II – 3 The salamander robot 6
III – Design: From C to Java 7
III – 1 C code 7
III – 2 Java code 8
III – 2.1 GraphicalUserInterfaceApplet 8
III – 2.2 Cpg 8
III – 2.3 OscilStructure 9
III – 2.4 Oscillators 9
III – 2.5 Square 9
III – 2.6 Graphs 9
III – 2.7 DrawGraph 9
III – 2.8 Axes 10
IV – Results: The Graphical User Interface 11
IV – 1 Settings panel 12
IV – 1.1 Sliders 12
Drive slider: Speed and gait regulation 12
Asymmetry slider: Turning 16
First oscillators drive slider: Changing the phase lags 17
IV – 1.2 Buttons 18
IV – 2 Oscillators panel 19
IV – 2.1 Green rectangles 19
IV – 2.2 Yellow rectangles 20
IV – 3 Graphs panel 21
IV – 3.1 Timer 21
IV – 3.2 Scale slider 21
IV – 3.3 Graphs 21
V – Problems Encountered 22
V – 1 Another developer 22
V – 2 C and Java 22
V – 3 Java itself 22
VI – Further Work 23
VII – Conclusion 24

I – Introduction

Within the framework of their study of animal locomotion, and more precisely of the transition from swimming to walking during vertebrate evolution, the team from the Biologically Inspired Robotics Group (BIRG) at EPFL developed a salamander robot capable of performing both walking and swimming gaits and smooth transition between the two.
The locomotion controller for this robot is a combination of a body central pattern generator (CPG) similar to a lamprey CPG for swimming and a limb CPG coupled to it to generate a walking gait. The salamander CPG is modelled as a chain of coupled nonlinear oscillators written in C code.
The aim of this project was to give a better visibility and accessibility to this research work by developing a Java applet for the controller. Researchers and scientists would this way be able to access the model from anywhere in the world through the internet and visualize the effects of different parameters on the behaviour of the oscillators through the user interface providing analysis graphs.
The first step was thus to carry the C code of the controller into Java code and check that it performed well. After that the graphical user interface had to be designed and developed and the different functionalities of the controller could be tested.

II – Background

II – 1 The real salamander

The transition from aquatic to terrestrial habitats has been one of the most important changes during vertebrate evolution. As an amphibian capable of both swimming and walking, the salamander is believed to be one of the modern animals closest to the first vertebrates that made this transition.
When it is underwater, the salamander uses a swimming gait similar to that of the lamprey, with the limbs folded backwards; it is based on a travelling wave of neural activity propagated from the head to the tail, which piece-wise constant wavelength corresponds to the length of the body. When it is on the ground, the salamander switches to a stepping gait with the phase relation of a trot; diagonally opposed limbs are in phase and laterally opposed limbs are out of phase. The trotting gait is based on S-shaped standing waves with nodes at the girdles; the stride length is optimized by coordinating the limbs with the bending of the body (see Fig. 1) [1, 2].

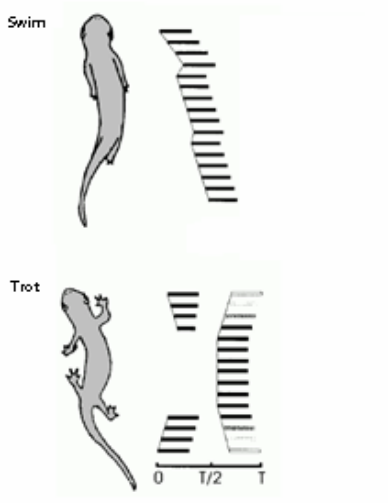

Figure 1 – Patterns of EMG activity during swimming and trotting [3]. During swimming we observe a travelling wave of muscle activity; the travelling wave has a piece-wise constant phase lag with small changes of wavelength at the girdles.
During walking the muscles in the trunk are synchronized, i.e. without phase lag.

Locomotion of the salamander is controlled by a network of neurons forming a central pattern generator (CPG). The neurons generate rhythmic output signals without receiving rhythmic inputs. The central nervous system of the salamander is very similar to that of the lamprey, a primitive fish which locomotor circuit has been extensively studied [2]. The main hypothesis is that the salamander's locomotor circuit is composed of a CPG controlling the body movements located all along the salamander's spinal cord, extended with limb CPGs located across from the limbs on the spinal cord.
Efficient neural network models of the salamander's locomotion circuit have been developed based on this hypothesis [4].

II – 2 The locomotion controller

This section is strongly based on the description of the model as presented in [5]; thus I will allow myself not to mention every time I quote.

II – 2.1 Nonlinear oscillator model

The locomotion controller for the salamander robot is modelled as a system of coupled nonlinear oscillators. Coupled oscillatory systems present the advantage that many of their properties only depend on the coupling between oscillators and not on their implementation. This allows to concentrate on the coupling itself rather than on the detailed neuronal mechanisms of rhythm generation.
Each oscillatory centre identified in the salamander's spinal cord is modelled by an amplitude-controlled phase oscillator. Such a model represents an ideal compromise between biological relevance and limited computing requirements such as being programmable on a microcontroller.
The nonlinear oscillator composing the building block of the model is implemented as follows:

$$\dot{\theta}_i = 2\pi\nu_i + \sum_j r_j\left(\alpha_{ij}\sin(\theta_j - \theta_i) + \beta_{ij}\cos(\theta_j - \theta_i)\right)$$
$$\dot{r}_i = a_i(R_i - r_i)r_i$$
$$x_i = r_i(1+\cos(\theta_i))$$

Where θ_i and r_i are the state variables of the oscillator, representing the phase and the amplitude respectively, ν_i and R_i determine the intrinsic frequency and amplitude and α_{ij} and β_{ij} are coupling weights determining how oscillator j influences oscillator i. For each oscillator, a signal x_i is extracted representing the amplitude of the burst produced by the centre.

In the salamander, the amplitude and frequency of oscillation increase with the level of tonic drive applied to the oscillators. When an increasing drive is applied to the CPG, three phases are distinguished: before the input drive reaches a first threshold no oscillation happens, between this threshold and a second one, oscillation frequency and amplitude increase with the drive and above the second threshold there's a saturation phase where the oscillations stop.

To model this effect, a piece-wise linear saturation function is introduced, that modulates the intrinsic frequency and amplitude ν_i and R_i according to the driving signal d_i:

$$\nu = g_\nu(d) = \begin{cases} c_{\nu,1} d + c_{\nu,0} & if \quad d_1 \le d \le d_2 \\ \nu_{sat} & otherwise \end{cases}$$

$$R = g_R(d) = \begin{cases} c_{R,1} d + c_{R,0} & if \quad d_1 \le d \le d_2 \\ R_{sat} & otherwise \end{cases}$$

Limb and body oscillators are provided with different saturation functions, with the limb oscillators systematically oscillating at lower frequencies than body oscillators for the same level of drive and saturating at a lower threshold (see Fig. 2).

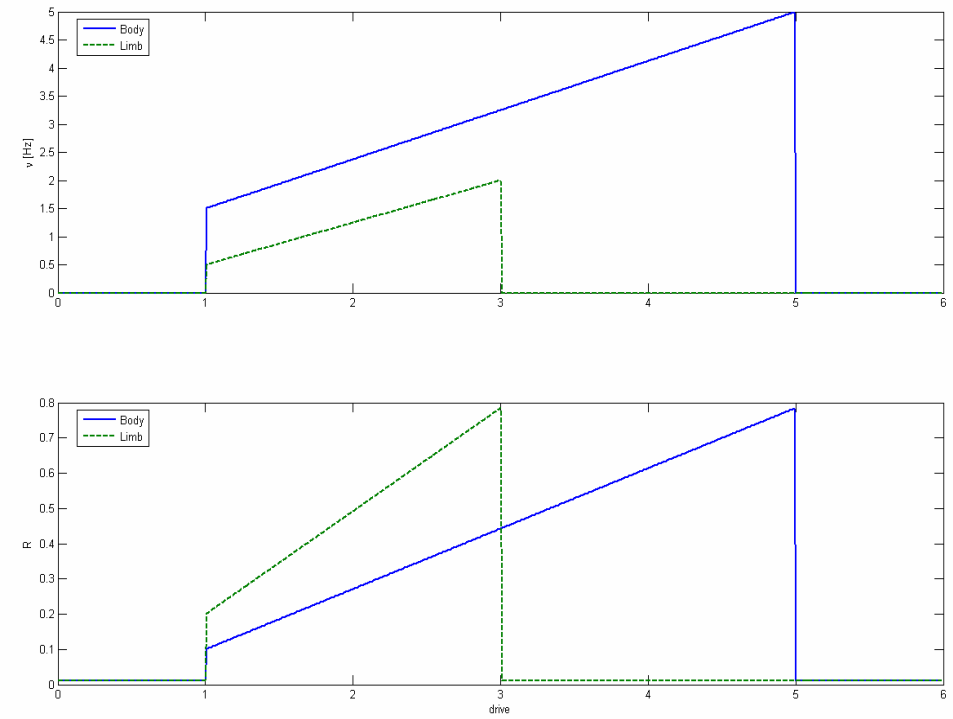

Figure 2 – Saturation functions for body and limb oscillators.
(Top) Intrinsic frequency. *(Bottom)* Intrinsic amplitude.

II – 2.2 Central pattern generator (CPG)

Central pattern generators are circuits that can generate rhythmic activity without rhythmic input. The complete CPG is composed of multiple oscillators coupled together with unilateral global coupling with limb intercoupling (see Fig. 3).

The body CPG is composed of a double chain of oscillators all along the 40 segments of the spinal cord, with closest neighbour coupling and the limb CPG is composed of four oscillators, one for each limb. The coupling from limb oscillators to body oscillators is unidirectional; left trunk oscillators receive inputs from the left forelimb, left tail oscillators receive inputs from the left hind limb and this is replicated on the right side.

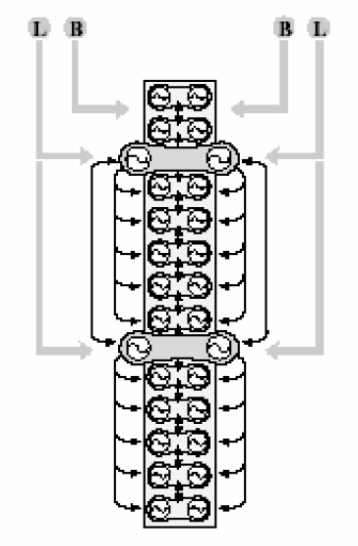

Figure 3 –Organisation of the CPG.

The coupling weights of the interconnections between two oscillators determine their phase difference (along with the difference of their intrinsic frequencies). They are defined as follows:

$$\alpha_{ij} = w_{ij}\cos(\phi_{ij}) \qquad \text{and} \qquad \beta_{ij} = w_{ij}\sin(\phi_{ij})$$

Where ϕ_{ij} is the phase difference towards which the oscillators *i* and *j* will converge and w_{ij} is the strength of the coupling.

The isolated body CPG produces travelling waves from head to tail with a wavelength of one body length; left and right oscillators oscillating in anti phase. The isolated limb CPG implements a trotting gait with left and right oscillators oscillating in anti phase and diagonally opposed oscillators in phase.

While swimming, the salamander keeps its limbs backwards along its body. In order to achieve this behaviour with the controller, the oscillators of the limbs need to be modified to converge to a specific phase, corresponding to a specific angle of the robot's legs. The differential equation for the phase of the limb oscillators thus becomes the following:

$$\dot{\theta}_i = 2\pi v_i + \sum_j r_j\left(\beta_{ij}^s \sin(\theta_j - \theta_i) + \beta_{ij}^c \cos(\theta_j - \theta_i)\right) + b_i \sin(\bar{\theta}_i - \theta_i)$$

Where $\dot{\theta}_i$ is the rest angle of the limb and *bi* is a gain that is set to zero when the drive is below the saturation threshold *d2* of the limb oscillators (when the salamander is walking), and set to 20 when it is above (when the salamander is swimming) .

II – 3 The salamander robot

The research on animal locomotion, and more specifically on lamprey-like locomotion, at the biologically inspired robotics group (BIRG) at EPFL leaded the researchers to build bio-inspired amphibious snake-like robots capable of crawling and swimming. Robots such as *Amphibot I* were very precious tools to investigate hypotheses about how central nervous systems implement locomotion abilities in animals [6, 7].
Current research focuses on the development of an amphibious salamander robot, capable of walking and swimming and performing a smooth transition between the two gaits. The robot is made of 9 segments (see Fig. 4). The head and six body segments are similar to the elements developed for *Amphibot II* [8]; they have a servo motor that enables horizontal motion with the previous segment. The two remaining segments are for the legs; each of them has two servo motors that enable independent rotation of 360° for the two legs.

Figure 4 – The real salamander robot.

The robot is still in its development stage and there is not much documentation available yet, thus I will not give any more details about it. However, a lot of research has already been done based on 2 and 3D mechanical simulations of the salamander robot. In particular, Ijspeert, Crespi and Cabelguen determined in [1] which CPG configuration was most likely to control salamander locomotion and investigated the mechanisms of entrainment between the CPG, the body and the environment. It is based on these simulation studies that they started designing the salamander robot to test whether the CPG models could be adapted to control a real robot.

III – Design: From C to Java

III – 1 C code

As I already mentioned, the first step of my project was to carry the C code of the controller to Java, in order to be able to develop it into an applet.
The C code is composed of three files: *cpg.c*, *cpg.h* and *pic_cpg.c*. The first and second ones implement all the functions needed to run the salamander CPG and the third one runs the CPG and saves its activity in different log files. The log files are then used to plot analysis graphs with Matlab® (see Figures 2 and 5).

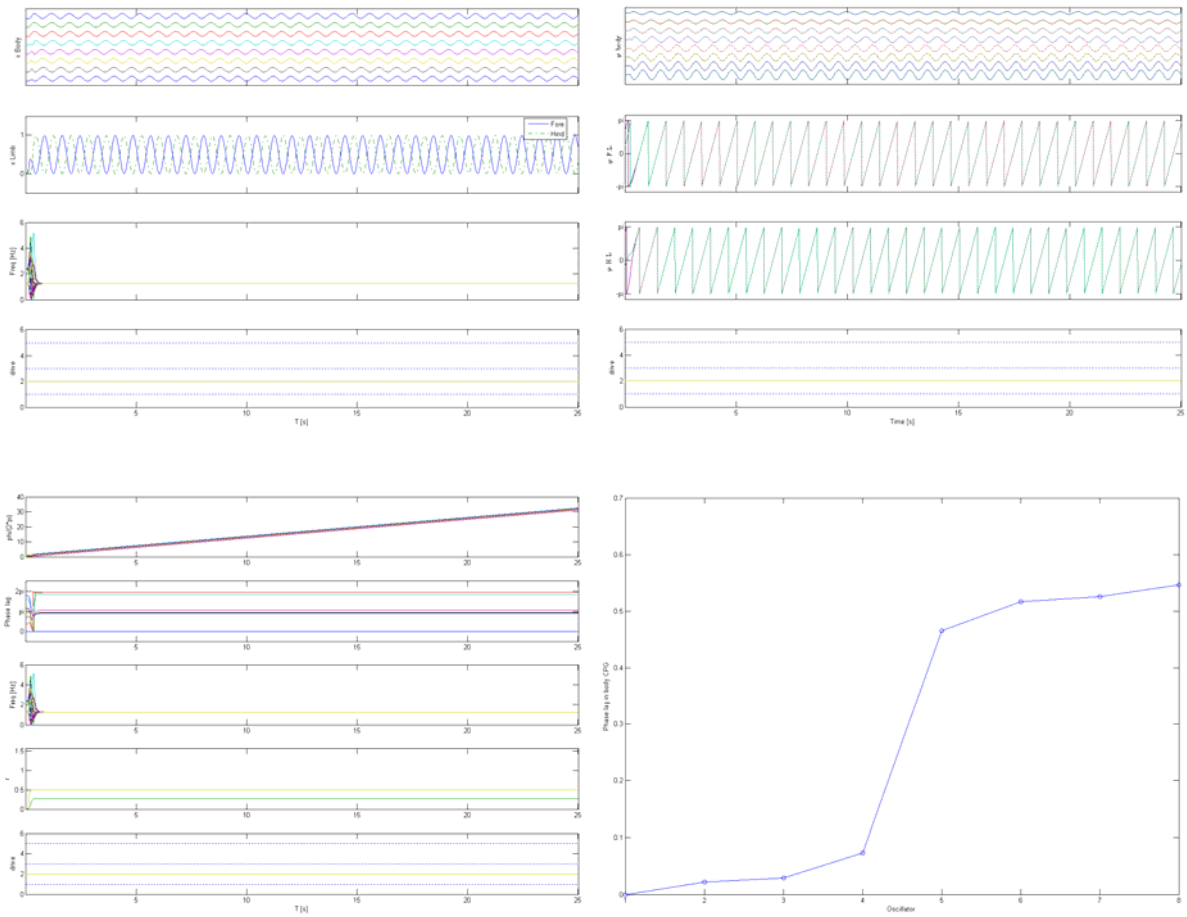

Figure 5 – Some of the analysis graphs plotted with Matlab® from the log files generated by the C program. *(Top left)* Activity of the CPG during walking. Only the oscillators on the left side of the salamander are represented. *(Top right)* Phase of the body and limb oscillators. *(Bottom left)* Evolution of the phase, phase lag of the oscillators, frequency of oscillations, amplitude and drive. *(Bottom right)* Phase lag of the oscillators in the body CPG.

III – 2 Java code

I structured the Java code in eight classes: *GraphicalUserInterfaceApplet.java*, *Cpg.java*, *OscilStructure.java*, *Oscillators.java*, *Square.java*, *Graphs.java*, *DrawGraph.java* and *Axes.java*. The following sections present a description and the UML diagram for each class; I omitted some of the attributes and operations (such as getters and setters) in the diagrams in order to preserve the readability of this document. To see the details of the implementation, please refer to the code and its documentation, available on the website (only from the EPFL network).

III – 2.1 GraphicalUserInterfaceApplet

This class initialises the graphical user interface (GUI) of the applet; the panels, labels, buttons and sliders are created and drawn.
It overrides the *javax.swing* and *java.awt stateChanged()* and *actionPerformed()* methods respectively for the sliders and the three buttons (play, stop and reset). In this class, the thread of the applet is created, initialised and run, and its *run()* method is implemented. The *main()* method of the *pic_cpg.c* program is implemented here as *computeOutputs()*; the drive signal is assigned to the oscillators and the saturation function and Euler integration are executed.
At this point, it is also possible to create the log files in the same way as in the C code.

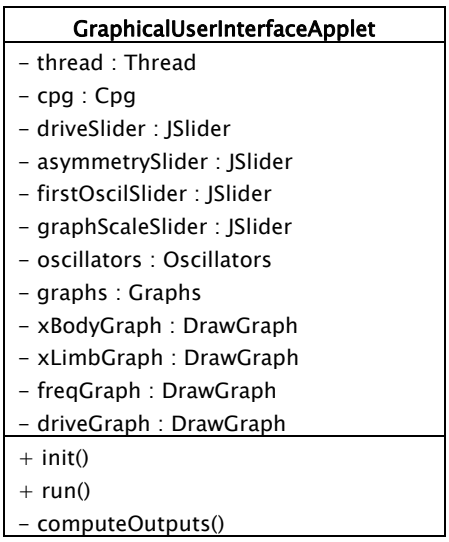

Figure 6 – GraphicalUserInterfaceApplet class diagram.

III – 2.2 Cpg

This class was directly carried from the *cpg.c* file. It implements the functions needed to run the salamander CPG, called in the *run()* and *computeOutputs()* method from the GraphicalUserInterfaceApplet class.
It also contains log making methods that allow the creation of the log files in the same way as in the C code. These methods need to be uncommented if the user wants to create logs.

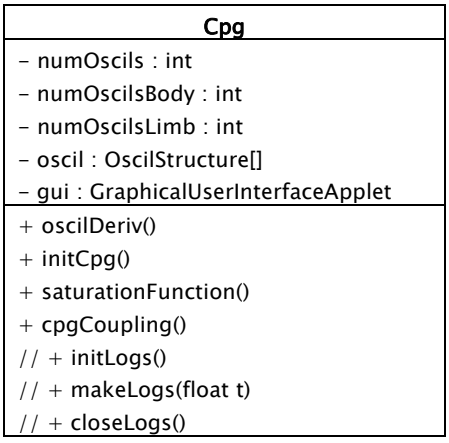

Figure 7 – Cpg class diagram.

III – 2.3 OscilStructure

This class implements the structure representing the oscillators composing the CPG and contains their parameters and variables.

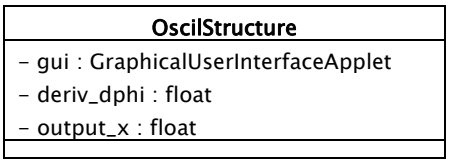

Figure 8 – OscilStructure class diagram.

III – 2.4 Oscillators

This class designs the panel that contains the graphical representation of the body and limb oscillators; it creates, initialises and displays the rectangles representing the input drive and the output values of the oscillators.

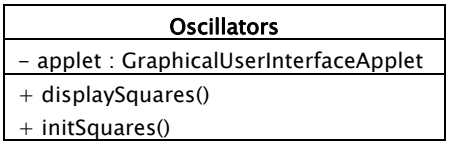

Figure 9 – Oscillators class diagram.

III – 2.5 Square

This class extend the *java.awt.Canvas* class and designs the graphical representation of the input drive and the body and limb oscillators' output values. The *update()* methods call the inherited *repaint()* method from *java.awt.Component*, that calls the *paint()* method from the class to compute and refresh the display of the oscillators.

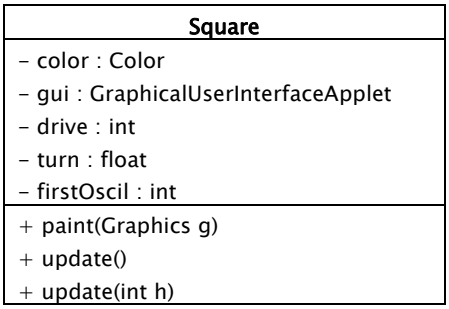

Figure 10 – Square class diagram.

III – 2.6 Graphs

This class designs the panel that contains the analysis graphs and the axes; it creates, initialises and displays them.

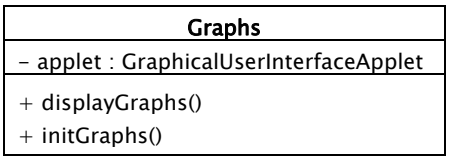

Figure 11 – Graphs class diagram.

III – 2.7 DrawGraph

This class extend the *java.awt.Canvas* class and designs the actual graph drawings. The *update()* method calls the inherited *repaint()* method from *java.awt.Component*, that calls the *paint()* method from the class to compute and refresh the display of the graphs.

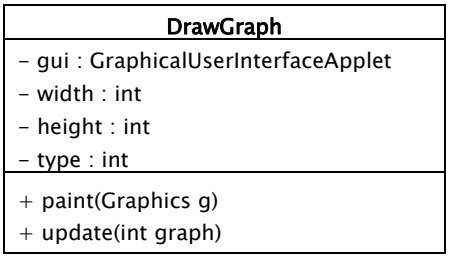

Figure 12 – DrawGraph class diagram.

III – 2.8 Axes

This class extend the *java.awt.Canvas* class and implements the axes of the graphs. The *update()* method calls the inherited *repaint()* method from *java.awt.Component*, that calls the *paint()* method from the class to compute and refresh the display of the axes, depending on the scale desired.

Axes
– gui : GraphicalUserInterfaceApplet – scale : float – width : int – height : int
+ paint(Graphics g) + update(float scale)

Figure 13 – Axes class diagram.

The Java code comprises some commented instructions and methods in the GraphicalUserInterfaceApplet and Cpg classes respectively. They allow the creation of log files in order to plot analysis graphs with Matlab® in the same way as with the C program; these graphs were useful to check that the Java program gave the desired outputs (the same as the C code – see Fig. 14).

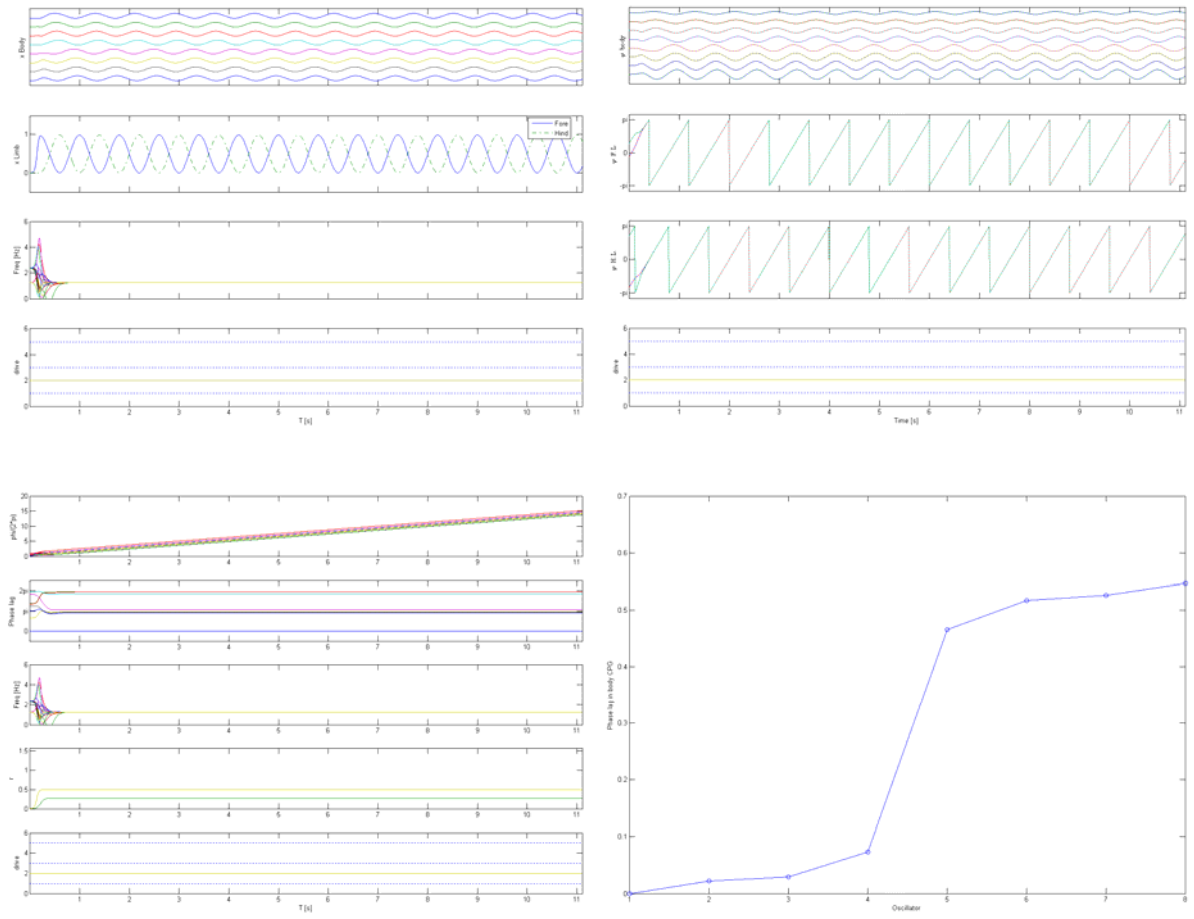

Figure 14 – Some of the analysis graphs plotted with Matlab® from the log files generated by the Java program. *(Top left)* Activity of the CPG during walking. Only the oscillators on the left side of the salamander are represented. *(Top right)* Phase of the body and limb oscillators. *(Bottom left)* Evolution of the phase, phase lag of the oscillators, frequency of oscillations, amplitude and drive. *(Bottom right)* Phase lag of the oscillators in the body CPG.

IV – Results: The Graphical User Interface

The GUI is divided into three main panels (see Fig. 15). The first panel is labelled *settings* and comprises three sliders to allow the user to set some parameters, and three buttons to play/restart, stop or reset the applet. The second panel is labelled *oscillators* and represents the activity of the body and limb oscillators of the salamander's CPG. The third and last panel is labelled *Graphs* and it contains the graph of the CPG activity.

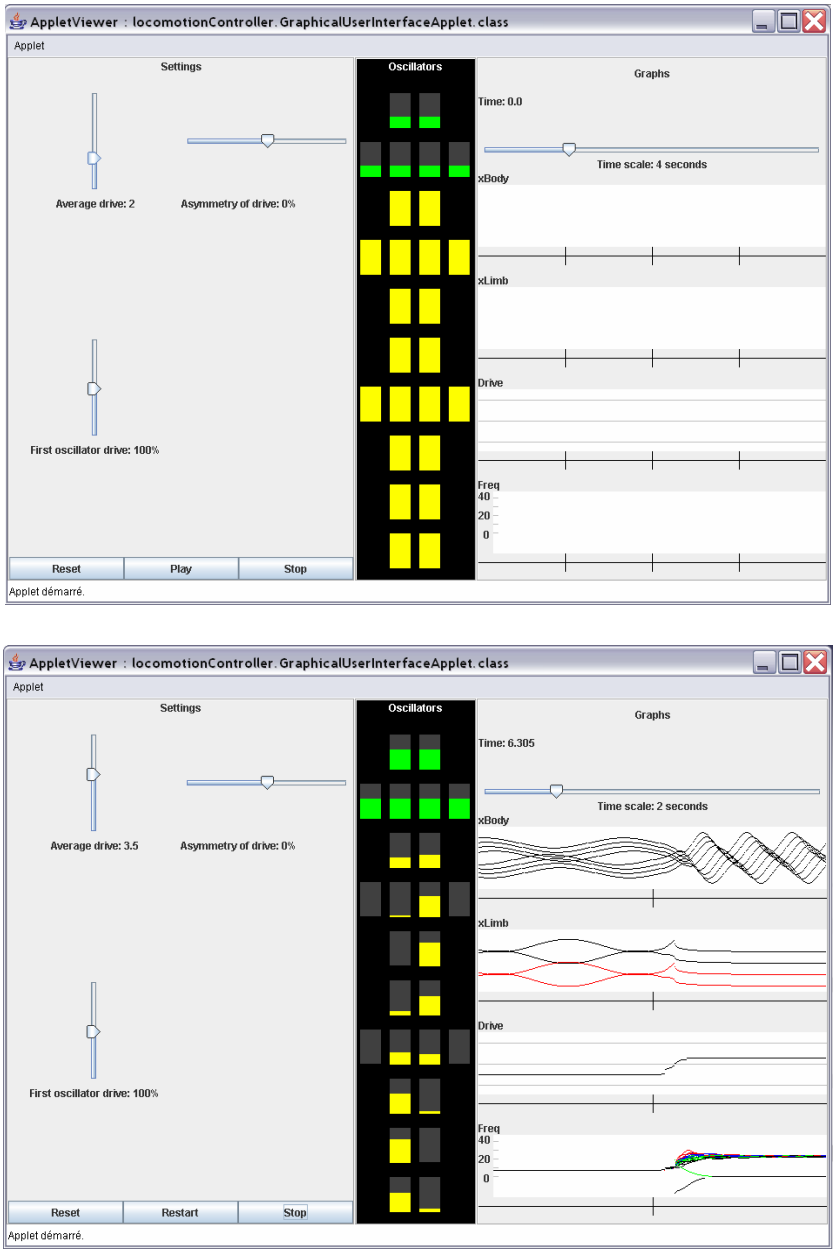

Figure 15 – Graphical User Interface. *(Top)* View at the start. *(Bottom)* In activity.

IV – 1 Settings panel

IV – 1.1 Sliders

The settings panel comprises three sliders to change the input drive applied to the body and limb oscillators of the salamander.

Drive slider: Speed and gait regulation

The first slider changes the value of the average drive for all oscillators; it regulates the speed of the oscillations and is responsible for the change of gait. Below a value of 1, the oscillators are all inactive; their output value is constant with negligible amplitude [5] (see Fig. 16).

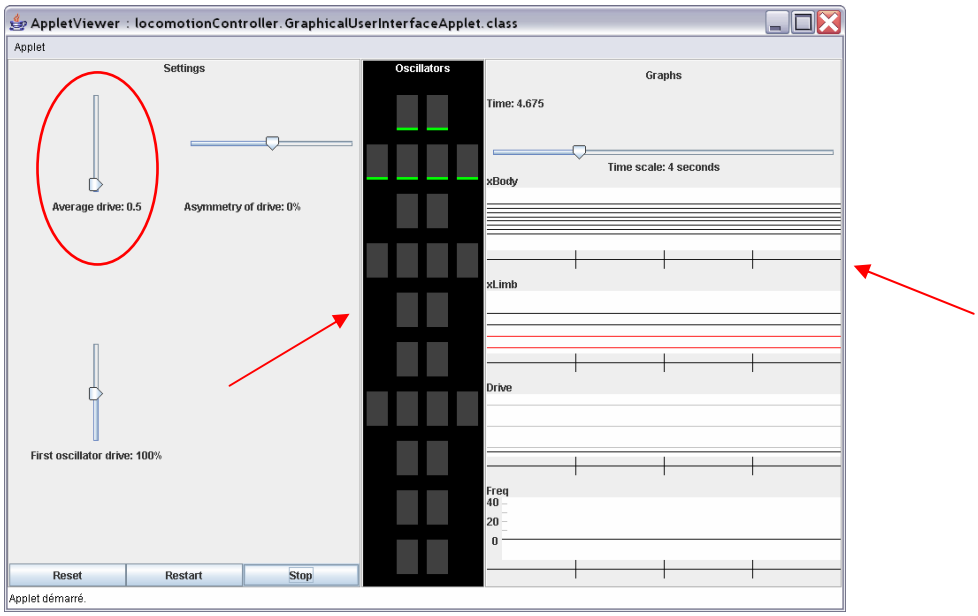

Figure 16 – GUI with all oscillators inactive.

Between 1 and 2.9, the oscillators from the limbs and the body are all active. Limb oscillators oscillate with the phase relation of a trot (with diagonally opposed limbs in phase and laterally opposed limbs out of phase – see Fig. 17) and the coupling from the limb to the body oscillators forces the body CPG to oscillate with an S-shaped standing wave. The body oscillators oscillate in anti phase between the rostral and caudal parts of the body and between the left and right side (see Fig. 18). In the salamander, the body-limb coordination is such that limbs are maximally turned forward when corresponding contralateral body segments are maximally contracted (see Fig. 19).

Increasing the drive leads to an increase of the frequency and amplitude of all oscillations and therefore to an increase of the walking speed, until the limb oscillators saturate and the transition to swimming occurs (see Fig. 20).

Figure 17 – Quadruped locomotion : trot.

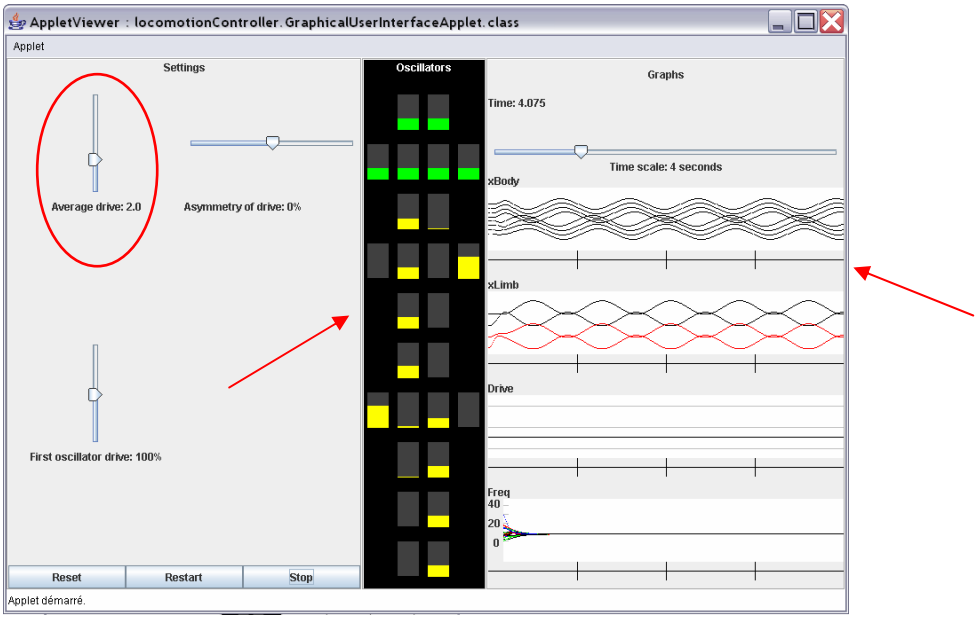

Figure 18 – GUI displaying a trotting gait.

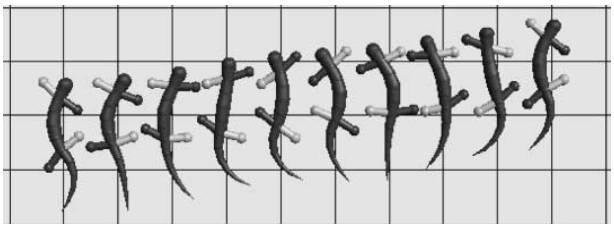

Figure 19 – Mechanical simulation of the salamander. Limbs in contact with the ground are drawn in white [4].

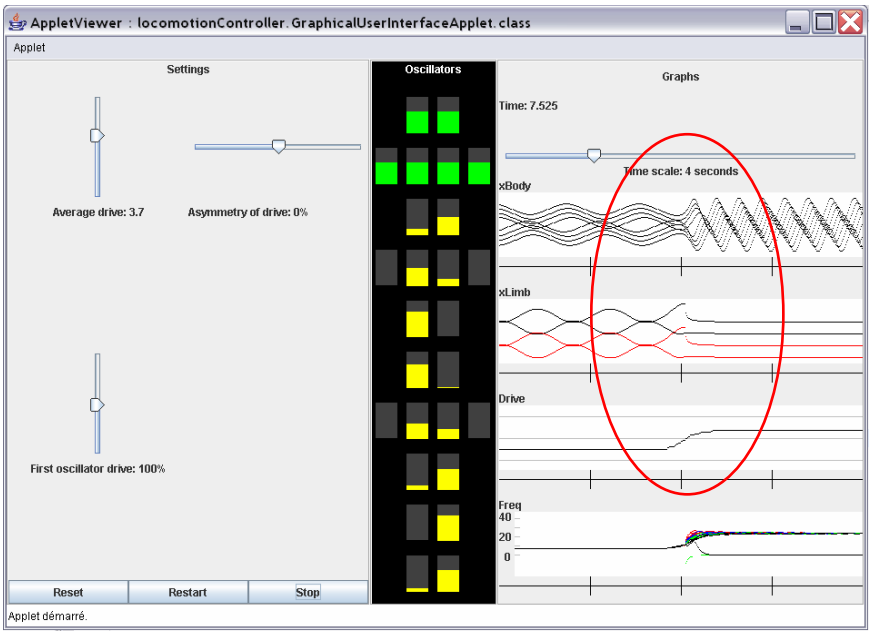

Figure 20 – Transition between walking and swimming.

When the drive is between 3 and 4.9, the limb oscillators are saturated (zero frequency and negligible amplitude) and thus they don't influence the body CPG any more; the body oscillators release travelling waves from head to tail with a wavelength equal to the length of the body (see Fig. 21). The travelling wave of body undulation allows the salamander to propel itself forward in water (see Fig. 22).
Increasing the drive leads to an increase of the frequency and amplitude of oscillations and therefore to an increase of the swimming speed, but the wavelength remains unchanged.

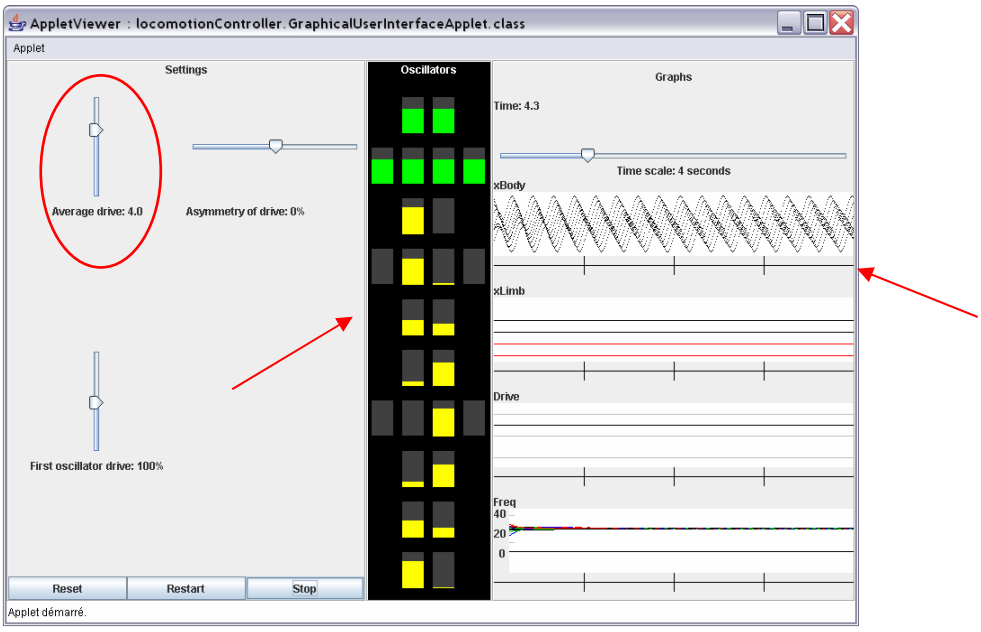

Figure 21 – GUI displaying a swimming gait.

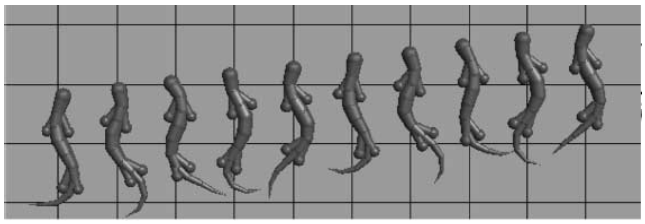

Figure 22 – Mechanical simulation of the salamander [4].

With a drive value above 4.9, all oscillators saturate and no oscillation is observed (see Fig. 23).

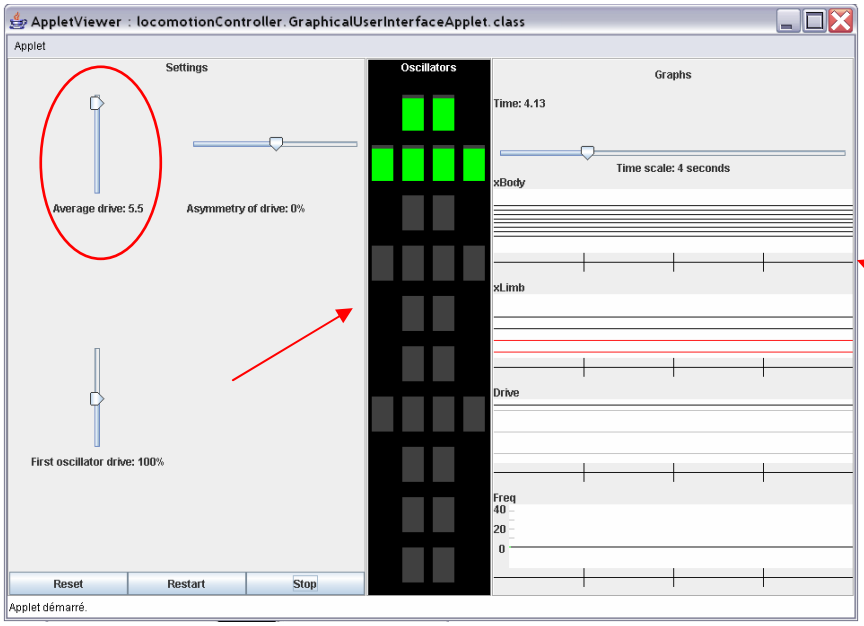

Figure 23 – GUI with all oscillators saturated.

Asymmetry slider: Turning

The second slider changes the value of the drive applied to the body oscillators; increasing it on one side and decreasing it on the other side. Applying asymmetrical drives between left and right sides of the body CPG induces turning during both walking and swimming (see Fig. 24 and 25). Changing the drive to an oscillator amounts to modifying its intrinsic frequency and its oscillation amplitude. When the difference in the left and right drive is not too large, the whole pool of oscillators keeps oscillating at a common frequency (i.e. the oscillators remain synchronized), and only an effect on the amplitudes can be observed; the side receiving higher drive will oscillate at higher amplitudes [5].

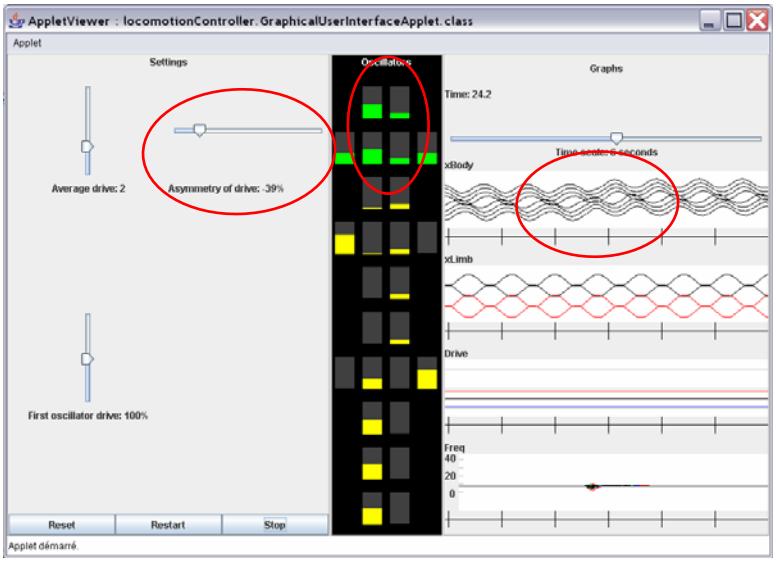

Figure 24 – Turning while walking.

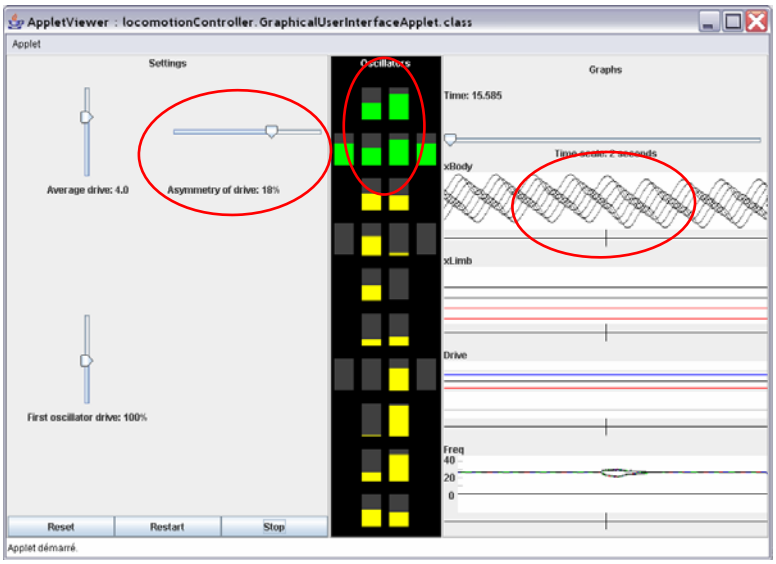

Figure 25 – Turning while swimming.

First oscillators drive slider: Changing the phase lags for swimming

The last slider in this panel allows changing the input drive of the head oscillators. When all body oscillators receive the same input, they produce a travelling wave with a wavelength corresponding to the length of the body. However, it is possible to modify this wavelength by applying a different strength of input to the first oscillators on the left and right side of the body, at the extremity of the chain. The resulting change of intrinsic frequency affects the phase lags along the whole chain (see Fig. 26). A higher drive leads to higher phase lags and lower wavelength, and a lower drive leads to lower phase lags [5]. If the drive difference is too large, the intrinsic frequencies of the head oscillators compared to the others in the chain become too large, the oscillators fail to synchronise properly and oscillations become irregular.
These effects are best viewed on the running applet; therefore I will not include any screenshots for it.

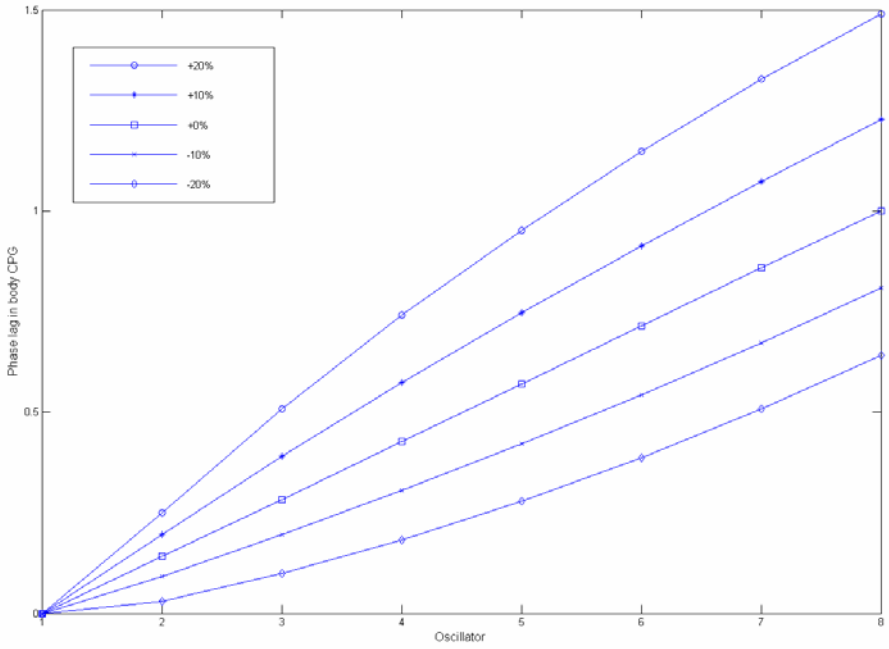

Figure 26 – Modification of the phase lag during swimming when the head input drive is modified.

IV – 1.2 Buttons

In addition to the sliders, the settings panel features three buttons to control the applet. When the applet is first initialised, the middle button displays the text *Play*; when it is pressed, a new thread for running the program is created and started. This leads to the CPG being created, initialised and started; the oscillators are set with the input drive value (computed from the three sliders), their outputs are computed and the display is updated. While the applet is running, the left and middle buttons become inactive.
The right button is always active. It displays the text *Stop* and when it is pressed it stops the applet without refreshing the display (the current state of the oscillators remains visible) and the left and middle buttons become active again, displaying the texts *Reset* and *Restart* respectively.
When the applet is stopped, the middle button displays the text *Restart*. Pressing it starts a new thread with the current value of the parameters (the slider values).
The left button, when pressed, resets the parameters to the initial ones, and resets the display of the oscillators and the graphs. The applet looks like when it was first started, the middle button displays the text *Play* again.

IV – 2 Oscillators panel

The middle panel comprises graphical representations of the input drive for the head, the body and the limb oscillators (the green rectangles), and of the oscillatory activity of the body and limb oscillators (the yellow rectangles).

IV – 2.1 Green rectangles

The green rectangles are set on two rows, two on the first row and four on the second. The rectangles on the first row represent the input drive applied to the head oscillators (the first two on the left and right side of the body, at the extremity of the chain). The rectangles on the extremities (left and right) of the second row represent the drive applied to the limb CPG; the two rectangles in the middle of the second row represent the drive applied to the body CPG (see Fig. 26). The size of all the rectangles varies depending on the values of the sliders from the settings panel (see Fig. 16, 18, 21, 23, 24 and 25).

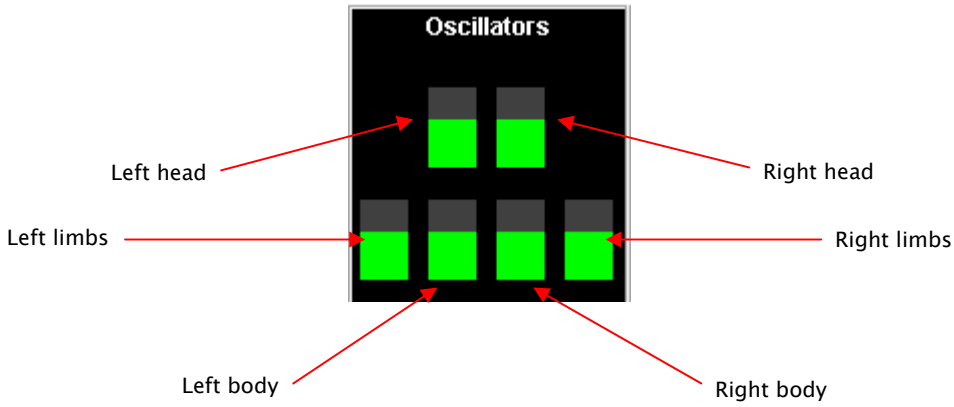

Figure 25 – Signification of the green rectangles.

IV – 2.2 Yellow rectangles

The yellow rectangles are disposed such as to represent the body and limbs of the salamander; the two middle columns represent the body oscillators, from head (top) to tail (bottom) and the rectangles on the outside (on the left and right side of the body) represent the four limbs (see Fig. 26). Their size represents the output signal of the oscillators (*output_x* in the Java code).

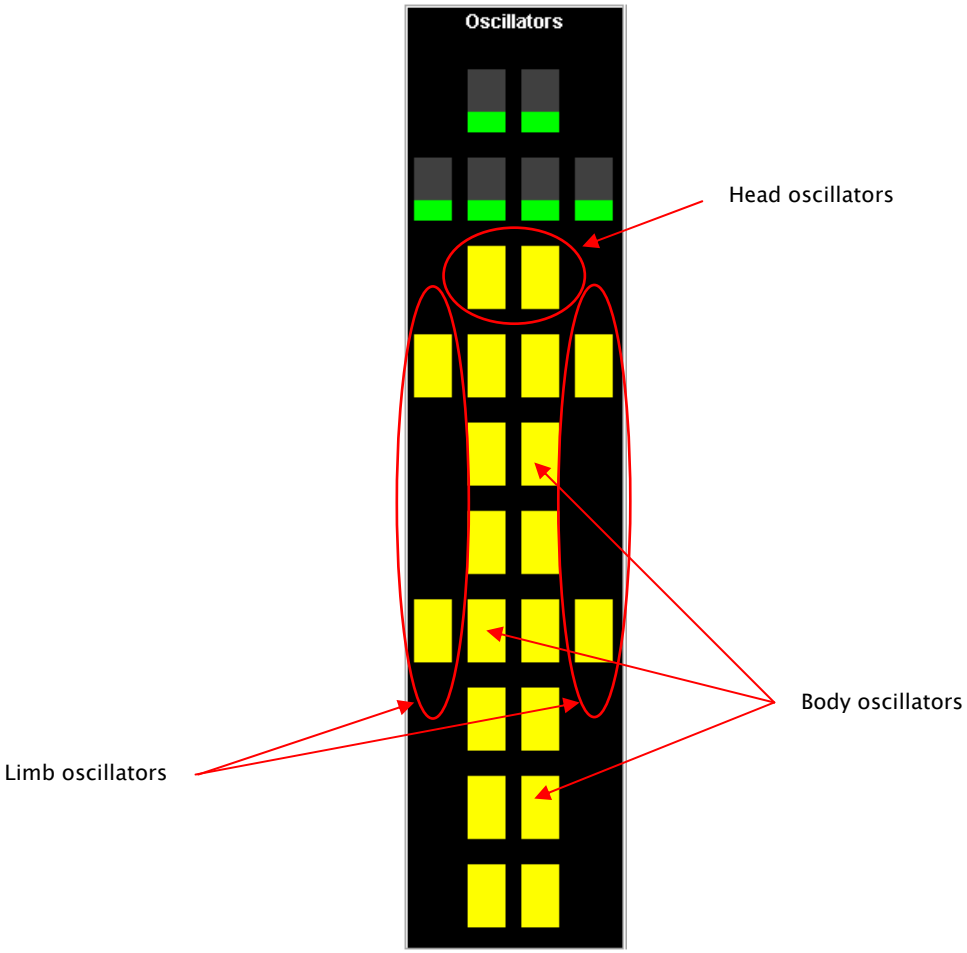

Figure 26 – Representation of the body and limb oscillators.

IV – 3 Graphs panel

The panel on the right of the applet displays the graphs useful to analyse the behaviour of the oscillators. It also features a timer to keep track of the simulation time and a scale slider that allows changing the scale of the graphs.

IV – 3.1 Timer

The timer displays the simulation time of the applet; it mainly depends on the occupancy of the Java virtual machine (JVM) that is required for the computations of the CPG and the display of the applet. Because the time is computed in relation to the JVM, it depends on its activity and does not pass with a constant speed.

IV – 3.2 Scale slider

The slider at the top of the graphs panel allows changing the scale of the graphs to see more or less details. It ranges between 2 and 10 seconds, with 2 seconds steps.

IV – 3.3 Graphs

The graphs panel comprises four graphs. The first graph, labelled *xBody*, plots the output signal of the oscillators; here the difference between the right and left oscillators is represented.
The second graph, labelled *xLimb*, plots the output of the four limb oscillators. The black curves represent the fore limbs and the red ones the hind limbs; they are drawn with a shift for a better visual effect.
The third graph, labelled *Drive*, plots the value of the input drive. The grey lines represent the threshold values 1, 3 and 5 for the oscillators' activity, the saturation of the limb oscillators (transition between walk and swim) and the saturation of the body oscillators, respectively. The average value of the drive (taken from the *drive slider*) is drawn in black; when the value of the asymmetry changes (with the *asymmetry slider*), the blue line represents the input drive for the right side of the body and the red line for the left side. When the drive value of the first oscillator changes (with the *first oscillator drive slider*), it is represented by a green line.
The last graph represents the frequency of oscillations. It is computed from the time derivative of the phase (*deriv_dphi* in the Java code) of the oscillators.

V – Problems Encountered

V – 1 Another developer

The first difficulty I encountered during this project was to understand a code developed by someone else. Indeed, it took me a while to realise for example that the indexes of the tables started at 1 (more natural for us humans) in the C program, and to make all the necessary changes to deal with it and let my table naturally start at the index 0 in Java.
It is never easy to understand the reasons underlying this or that choice of structure or implementation, and then decide whether it should be kept this way or changed.

V – 2 C and Java

C and Java are fundamentally different programming languages. Firstly C is not object oriented; it doesn't feature basic concepts such as inheritance and encapsulation that are extensively used in a language like Java. The other main difference is that Java uses dynamic allocation of the memory at execution time instead of static allocation at compilation time for C. Part of the program therefore had to be redesigned, while respecting the author's initial ideas and broad lines.

V – 3 Java itself

Not being very familiar with Java at the start of this project, I had to come to grips with it and start from scratch. Keeping the general concepts implemented in the C program while respecting the requirements of Java has not proved too hard but some fundamental decisions had to be made. The most important change in the design of the program was to set all the variables as *private* and generate getters and setters to handle them; this way, their use is under better control and more secure. Another noteworthy change I made was to avoid the use of static variables as often as I could when it was not necessary.

VI – Further Work

The applet could always be improved in many ways; I will point out here four improvements that I think would be worth implementing.

At the moment, all the calculations and displays are executed in one thread. I tried separating the computation of the oscillators' outputs and their display in two different threads but didn't manage to and as time was going by I had to concentrate on more matters. **Multi-threading** would probably lead to an easier implementation of real time running, described in the next section.

As I mentioned before, the timer on the right panel of the applet displays the simulation time, which is different to the real time and in addition does not pass in a constant manner. Making the applet **run in real time** would be much more sensible and user-friendly. I also tried to implement this feature but again didn't manage to, even after trying different methods. My failure was mostly owing to not being able to deal with the unpredictable display refreshment performed by the JVM.

The display of the graphs could be improved by performing **double-buffering**. Hopefully I will soon be able to put an upgraded version of the applet available on the website, which will implement this.

The icing on the cake for this applet would be to **display a schematic salamander body shape,** showing the position of the salamander body parts in relation to the activity of the oscillators. It could for example be located in the empty space on the settings panel and in this way not load down the GUI.

VII – Conclusion

Looking back on the aims of the project, we see that the main goals have been achieved. The developed graphical user interface successfully displays the behaviour of the salamander CPG and the effects of some parameters on the oscillators can be observed. The design is rather simple and intuitive and the visual outputs are easy to interpret.

The time investment necessary for me to get to grips with Java detracted the next planned step of the project, namely the study of the model. I would have been very interested in studying the model of the controller and its similarities and differences with the real animal, and the few papers I read about some ongoing research already made my mouth water. Yet time sometimes goes faster than one thinks (especially when sitting in front on a computer!) and I am quite happy with the outcome of the project that allowed me to learn a lot about Java.

Bio-inspired systems are a field that I would be interested in studying further, both from a modelling and biological point of view and I hope to be able to keep the orientation of my studies and later work directed towards this goal.

Acknowledgements

I would like to thank Professor Auke J. Ijspeert for his time and his regular feedback on this project. I would also like to acknowledge his assistant and supervisor for this project, Alessandro Crespi for his constant support and valuable help with Java.
Thumbs up too to the whole BIRG team for their daily enthusiasm and evident pleasure in their work and will to share it!

References

[1] Ijspeert A.J., Crespi A. & Cabelguen J-M., *Simulation and Robotics Studies of Salamander Locomotion: Applying Neurobiological Principles to the Control of Locomotion in Robots*, Neuroinformatics, 3 (3), 171-196, 2005.

[2] Ijspeert A.J. & Cabelguen J-M., *Gait transition from swimming to walking: investigation of salamander locomotion control using nonlinear oscillators*, Proceedings of Adaptive Motion in Animals and Machines, 2003.

[3] Devolve I., Tiaza B. & Cabelguen J-M., *Epaxial and Limb Muscle Activity During Swimming and Terrestrial Stepping in the Adult Newt, Pleurodeles waltl*, Journal of Neurophysiology, 78, 638-650, 1997.

[4] Ijspeert A.J., *A connectionist central pattern generator for the aquatic and terrestrial gaits of a simulated salamander*, Biological Cybernetics, 84 (5), 331-348, 2001.

[5] Ijspeert A.J., Crespi A. & Cabelguen J-M., *Supplementary material to "Gait transition from swimming to walking in a salamander robot using central pattern generators"*.

[6] Crespi A., Badertscher A., Guignard A. & Ijspeert A.J., *Amphibot I: an amphibious snake-like robot*, Robotics and Autonomous Systems, vol. 50 (4), 163-175, 2004.

[7] Crespi A., Badertscher A., Guignard A. & Ijspeert A.J., *An amphibious robot capable of snake and lamprey-like locomotion*, Proceedings of the 35th international symposium on robotics, 2004.

[8] BIRG webpage, Amphibot: http://birg.epfl.ch/page53468.html