



BIOLOGICALLY INSPIRED
ROBOTICS GROUP (BIRG)

Semester Project : Synchronization of movements of a real humanoid robot with music

Patrick Amstutz¹

February 17, 2007

¹Department of Communication and Computer Science, Swiss Federal Institute of Technology, Lausanne, Switzerland

Contents

1	Introduction	4
1.1	Goals of the project	4
2	Humanoid Robot	5
2.1	Actual implementation and global CPG architecture	5
2.2	Mathematical representation of Dynamical Systems	6
2.3	Study of Dynamical Systems with Matlab	8
2.3.1	Phase shift during discrete movement, online modulation of g_i parameter	9
2.4	Phase shift during amplitude changes, online modulation of μ_i parameter	11
3	Tempo detection algorithm	13
3.1	Scheirer Method	13
3.1.1	Step 1: Filterbank	13
3.1.2	Step 2: Smoothing	14
3.1.3	Step 3: Diff-Rect	14
3.1.4	Step 4: Comb Filter	14
3.1.5	Matlab implementation	15
3.1.6	Further work	15
3.2	Aubio C Library	15
3.2.1	Beat tracking algorithm	15
3.2.2	Onset Detection	16
3.2.3	High Frequency Content (HFC)	17
3.2.4	Complex Domain Onset Detection	17
3.2.5	Tempo Analysis	17
3.3	Beat tracking implementation	17
4	Webots Implementation	18
4.1	Original implementation of hoap-2 performing a drumming task .	19
4.2	Implementation of Webots collision detection	20
4.2.1	The physic plugin	20
4.2.2	ODE interface library functions	21
4.2.3	Integration of tempo detection	23
4.2.4	Message	23
4.2.5	The Supervisor	25
4.2.6	Video generation	25
5	Music generation with Matlab	26
5.1	Data from Webots simulation	26
5.2	Matlab Script	26

6	Merging Video and Music	26
6.1	VirtualDub	26
6.1.1	VirtualDub Scripting	27
6.1.2	Script Generator	27
7	Conclusion	28
8	Acknowledgment	28
A	Processing Schema	29
B	Dynamic Hebbian learning in adaptive frequency oscillators	30
B.1	Mathematical Aspects	30
B.2	Time needed for synchronization with coupled oscillator's frequency	30
B.3	Applying the Dynamic Hebbian learning in adaptive frequency oscillators to the dynamical system of the CPG	31
C	Collision detection issue	35
D	Screenshots	36

1 Introduction

The Biologically Inspired Group (BIRG) at the Swiss Institute of Technology Lausanne is working on the computational aspects of movement control, sensorimotor coordination, and learning in animals and in robots. One of the actual research topics is humanoid robotics. The present project is linked with the research leaded by Sarah Degallier concerning the online generation of trajectories in humanoid robots [2]. The actual demonstration model takes the form of a drumming robot. The control is done exclusively on the top half of the robot's body; both arms actually. The system presented, which uses a Hoap-2 humanoid robot, allows the superposition, and switch between, discrete and rhythmic movements. This example is particularly well suited to demonstrate these two aspects of movement and will be seen in more details in section 2.

Music is based on rhythms and thus represents a natural substrate for experiencing rhythmic movements, with high pedagogic quality. The drumming task has the particularity to offer simple discrete and rhythmic movements while being realizable from a mechanical point of view.

The online generation of trajectories in humanoid robots remains a difficult problem. While excellent progress has been made on designing efficient controllers for trajectory following in humanoid robotics, the problem of generating the trajectories themselves is still a complex, unsatisfactory solved problem. One of the main difficulties is that the trajectory generation problem is highly task-dependent and often requires extensive knowledge of the task to be solved.

The approach uses nonlinear dynamical systems, i.e. systems of differential equations, for generating trajectories online and in real time. Dynamical systems can be designed to have interesting attractor properties which makes them well-suited for trajectory generation. These properties particularly include: intrinsic robustness against small perturbations, possibility to change parameters on the fly (i.e. to do online modulation) and possibility to synchronize with external signals.

1.1 Goals of the project

The goals of the project are the following:

- Study the actual trajectories generation system
- Find an approach for adding sound capabilities for Webots simulations' videos
- Implement collision detection for visual feedback of drumming task
- Synchronize the robot drumming task with music: investigate possible solutions for extracting tempo information from music.

Different nonlinear dynamical systems are firstly described and tested against parameters change. The possibility of synchronization with external signal is then explored.

The modeling of the simulation is performed in the professional mobile robot simulator Webots [4]. The drumming robot is represented by a model of Hoap-2 humanoid robot. Webots do not offer the native capability to include sound in the simulation movies. The first objective of the project is to find a way to add sound effect in correlation with the events from the movies. In our case, the goal is to add the audio track of the drumming task. To perform this, collision detection is first added to the simulation using ODE plugin [4]. The collisions represent the contact between the drums and the sticks, thus the piece of music. Sound file is then generated using a Matlab script. The collision detection is also used in modifying instrument’s texture for debugging and visual feedback purposes.

Approaches for extracting tempo information from music is then investigated. The main point of this part is to design an approach that allows the robot to synchronize with the tempo of some external music source. Tempo extraction is known to be hard problem and as I had no background on the domain, this appeared as the most challenging part of the project. Firstly, two different algorithms are found and evaluated, and then one of them is implemented in the Webots simulation. The tempo extracted influences the robot’s rhythmic speed.

The last part of the project is to find solutions to automate the “from simulation to final video with music” process, in other words, reduce and simplify as much as possible the needed steps to produce the final video with sound, from the soundless video with textual information about the simulation’s collision.

2 Humanoid Robot

In this section, the actual implementation of the robot performing drumming task is reviewed. First the global system is described followed by detailed description of the dynamical systems used. The dynamical systems are then tested with Matlab simulation concerning the modulation of parameters and phase shift. Finally, different solutions are proposed and validated against this issue.

2.1 Actual implementation and global CPG architecture

The mechanism of control of a generic CPG is illustrated on figure 1. The system is built on the hypothesis that complex movements can be generated through the superposition and sequencing of simpler motor primitives implemented as dynamical systems. In particular, this system is made of sets of motor primitives which implement dynamical goal directed and also rhythmic movements.

The system is implemented on Webots simulation as well as on real Hoap-2 robot. It is a 25 DOFs humanoid robot made by Fujitsu. The system uses 8 of the 25 DOFs of the robot, that is 4 DOFs in each arm : 3 in the shoulder and 1 in the elbow. Figure 2(a) shows a schematic view of the controlled DOFs of the Hoap-2 robot. The others DOFs remain fixed to an appropriately chosen value during the task. For each controlled DOF, a generic CPG presented in

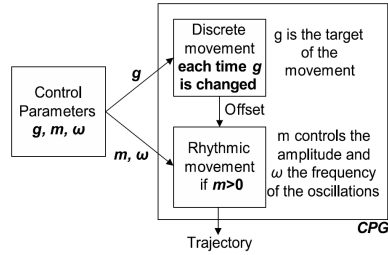


Figure 1: A generic CPG. Modes of movement are switched on and off by parameters m and g . Discrete movement is incorporated to the final trajectory as an offset to the rhythmic movement. Trajectory is modulated by particular choices of m and g .

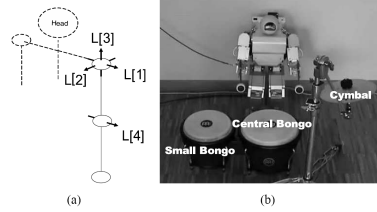


Figure 2: (a) Schematic view of the controlled DOFs of the Hoap-2 left arm. The corresponding axes of rotation are also represented, (b) Picture of the real humanoid Hoap-2 robot sitting in front of 3 instruments : a central drum, a small drum and a cymbal.

section 2.2 is used. The CPGs network constitutes the controller generating the trajectories which are used as input for the PID controllers of each joint. The overall system architecture is depicted in figure 3. A arbitrary score matrix M is transformed onto a time-varying parameter vector, $\tilde{\alpha}$, that controls the parameters of the CPGs network that generate drumming trajectories in real time. More precisely, for each DOF i and for each beat, this vector specifies the goal g_i and the amplitude of the movement, controlled by μ_i . Desired trajectories \vec{x} of each DOF are obtained by integrating the CPGs dynamical systems. These trajectories are used as input for the PID controllers of each joint and result in the actual trajectories \vec{x} .

Hopf oscillators of the CPGs are bilaterally coupled, those couplings being illustrated by right-left arrows on figure 3. One CPG's rhythmic component is also coupled with an external clock using the same type of coupling. This clock is an Hopf oscillator of parameters μ_{clock} and ω_{clock} .

2.2 Mathematical representation of Dynamical Systems

Different representations of the mathematical equations have been studied. The first one is the original form presented by Degallier and al. The equations are under Cartesian form and the discrete part is included through hierarchic way. The CPG for a given DOF i is divided in two subsystems, one generating the discrete part of the movement and the other generating the rhythmic part.

- The discrete part is given by the two signals Y and V . The equations are the following:

$$\dot{y}_i = v_i \quad (1)$$

$$\dot{v}_i = \frac{-b^2}{4} (y_i - g_i) - b v_i \quad (2)$$

Eqs 1 and 2 describe a discrete motion whose solution y_i converges asymptotically and monotonically to a goal g_i with speed of convergence controlled

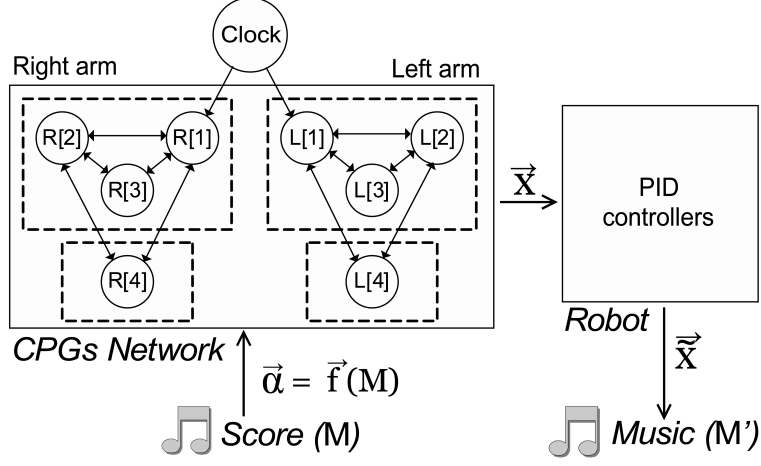


Figure 3: (a) Schematic view of the controlled DOFs of the Hoap-2 left arm. The corresponding axes of rotation are also represented, (b) Picture of the real humanoid Hoap-2 robot sitting in front of 3 instruments : a central drum, a small drum and a cymbal.

by b . It generates the discrete movement toward g_i . As shown in figure 4, each time g_i is changed, the system will be attracted by the new goal g_i and modify the resulting position y_i , generating a discrete movement toward g_i .

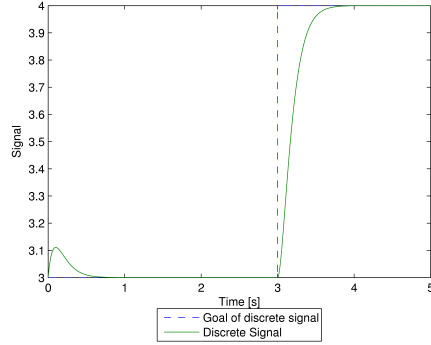


Figure 4: Progression of discrete part of the signal. Trajectory is modulated by particular choices of g .

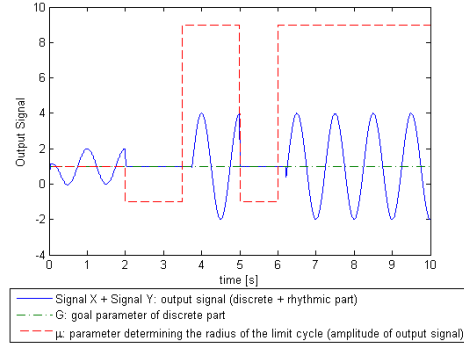


Figure 5: Progression of rhythmic part of the signal. Trajectory is modulated by particular choices of μ .

- The rhythmic part is described by a Hopf oscillator, i.e. by the following system:

$$\dot{x}_i = a(\mu_i - r_i^2)(x_i - y_i) - \omega_i z_i \quad (3)$$

$$\dot{z}_i = a(\mu_i - r_i^2)z_i + \omega_i(x_i - y_i) \quad (4)$$

where $r_i = \sqrt{(x_i - y_i)^2 + z_i^2}$.

Eqs 3 and 4 describe a Hopf oscillator where μ_i controls the amplitude of the oscillations, ω_i is the oscillator intrinsic frequency and a controls the speed of convergence to the limit cycle. The output x_i of the system has an offset given by y_i which is the state variable of the discrete system.

Rhythmic motion can be switched on or off by simply setting μ_i to a positive or a negative value respectively. Moreover, the amplitude of the movement is specified by μ_i and its frequency by ω_i . Different modifications of the parameters are shown in figure 6. Parameters used by the simulations in this section are the following: $a = 100, b = 20, \omega = 2\pi$.

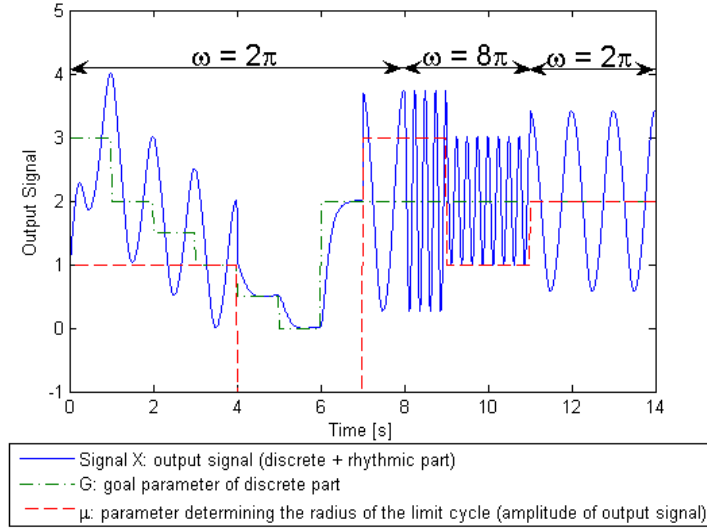


Figure 6: Progression of output signal (hierarchical / Cartesian coordinates) subject to discrete and rhythmic movements. Trajectory is modulated by particular choices of g , μ_i and ω_i .

2.3 Study of Dynamical Systems with Matlab

A particularity of dynamical systems is that, instead of encoding a trajectory explicitly, they encode a whole state space and its time evolution. This means that the system must be integrated over time to generate the trajectory (i.e. the trajectory can not be instantaneously extracted out of the system), and that it encodes more than just the trajectory, since it also encodes how the trajectory evolves after a transient perturbation. Thus in order to analyze the effect of discrete movement with the preceding dynamical system we need to run integration experiments using Matlab. The numerical integration used for calculating the trajectories is the “Euler Integration”. Euler integration is simply derived from equations for the derivatives of the position $x(t)$ and

velocity $v(t)$ of an object:

$$v(t) = \frac{dx(t)}{dt} = f(x(t))$$

$$x(t + \Delta t) = x(t) + f(x(t)) \cdot \Delta t$$

with $f(x(t))$ being the derivative of x at time t and Δt the integration step. As the trajectory resulting from the numerical integration depends directly on the initial conditions, many have been tested during simulations.

2.3.1 Phase shift during discrete movement, online modulation of g_i parameter

One possible issue with the actual dynamical system is the possible phase shift to the rhythmic part of the CPG when adding a discrete movement to the trajectory. Indeed, each time g_i is changed, the offset of the system is modified and make it tend to the new goal g_i .

As we can see on figure 7, the phase of the output signal changes during the discrete movement, this implies a phase shift when the system reaches the new goal g_i . To emphasize this behavior, a Hopf oscillator having the same properties as the initial system is plotted at both discrete start level and discrete goal level.

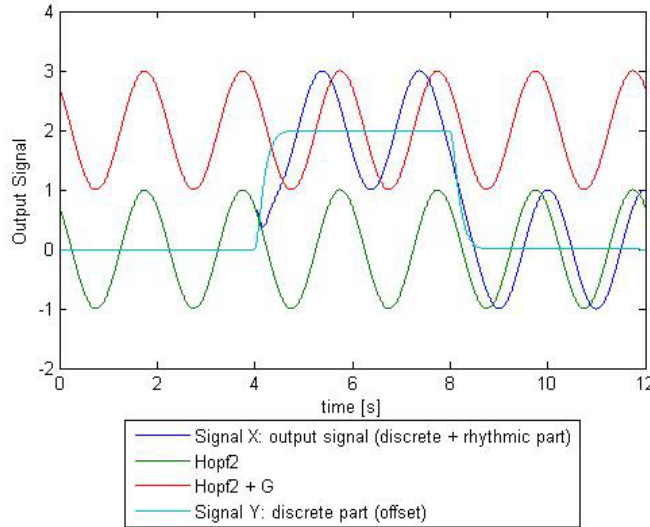


Figure 7: Progression of output signal (hierarchical / Cartesian coordinates) subject to discrete movements. Trajectory is modulated by particular choices of g . We clearly see the final phase shift between initial and final signal. Hopf oscillator signal as reference.

To avoid such problem, an additive way to combine rhythmic and discrete part of the system is studied. The discrete part does not change, but the rhythmic part gets rid of the direct influence of discrete part. This new approach

can be seen as a way to “preserve” the rhythmic part from the influence of the discrete one during the numerical integration while adding the discrete offset afterward. We can hypothesize that such a system, which is given by the following equations, will be able to deal with phase shift.

$$\dot{x}_i = a(\mu_i - r_i^2)x_i - \omega_i z_i \quad (5)$$

$$\dot{z}_i = a(\mu_i - r_i^2)z_i + \omega_i x_i \quad (6)$$

The combination of both rhythmic and discrete parts is done externally to the equations by simply summing the two signals X and Y :

$$\text{output signal} = X + Y$$

This new system, called “additive system”, is numerically integrated and the resulting trajectories are shown in figure 8. By analyzing the data, we clearly see that the system presents no phase shift anymore due to discrete movement. This result is in agreement with the preceding assumptions.

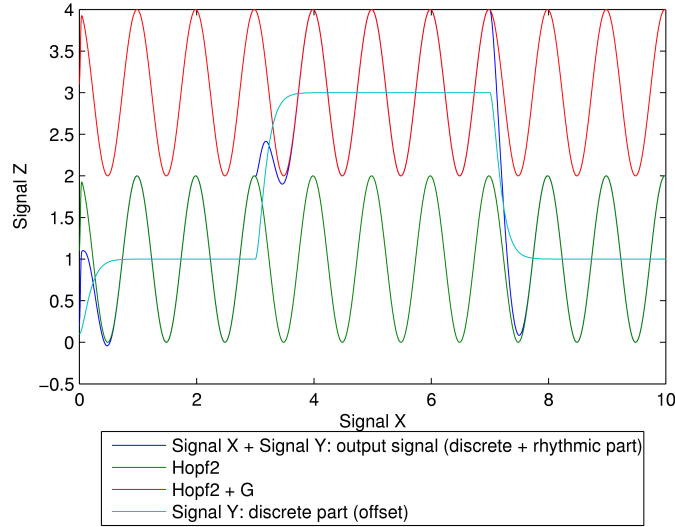


Figure 8: Output signal.(additive / Cartesian coordinates) subject to discrete movement with Hopf oscillator. Trajectory is modulated by particular choices of g .

The next step is nothing more than a mathematical coordinate system conversion. The equations of the rhythmic part are converted into polar coordinates using the well known trigonometric relations $x = r \cos \theta$ and $z = r \sin \theta$:

$$\dot{r}_i = a(\mu_i - r_i^2)r_i \quad (7)$$

$$\dot{\phi}_i = \omega \quad (8)$$

Experiments show that the transition to polar coordinates does not change the properties of the system.

The following conclusion can be stated from the preceding numerical simulations. The phase shift induced to the movement by the discrete part is successfully corrected by using the additive approach. The computation of rhythmic part without intrinsic influence of the goal's offset allows the phase not to be wrongly influenced. The experiments have not shown any influence from the parameter b , representing the speed of convergence to goal g_i .

The "additive" system is used in the Webots implementation.

2.4 Phase shift during amplitude changes, online modulation of μ_i parameter

As we have seen, μ_i controls the amplitude of the oscillations. The oscillator contains a bifurcation from a fixed point (when $\mu_i < 0$) to a structurally stable, harmonic limit cycle with radius $R = \sqrt{\mu_i}$ for $\mu_i > 0$. Rhythmic motion can be switched on or off by simply setting μ_i to a positive or a negative value respectively. Simulation have been used to evaluate the influence of such changes on the phase, and deduce possible phase shift when performing online modulation on μ_i .

Firstly different values $\mu_i > 0$ have been tested. As we only use strictly positive μ_i parameters, the oscillator always presents a limit cycle behavior. By modulating μ_i , we can evaluate the effect of the amplitude's changes on the phase.

As we can see in figures 9 and 10 the modulation of the μ_i parameter among positive ones does not affect the phase of the oscillator.

Secondly, the parameter μ_i is changed from positive values ($\mu_i > 0$, limit cycle) to negative ones ($\mu_i < 0$, fixed point). Except for μ_i , the parameters do not change from the preceding simulation.

The results are represented in figures 11 and 12. Contrary to the preceding simulation with only strictly positive μ_i values, the exploration of positive and negative values together has as a consequence to induce a phase shift. The experiments have not shown any influence from the parameter a , representing the speed of convergence.

The experiments, done with Cartesian and Polar coordinates version of additive system, show the same properties; the change of the μ_i parameter from positive to negatives ones has for consequence to produce phase shift. The hierarchic system shows both phase shifts after discrete movement and amplitude changes.

In conclusion to phase shift experiments, we can say that this effect is unavoidable when the rhythmic part is deactivated ($\mu_i < 0$) and reactivated ($\mu_i > 0$) at random time during the simulation. When rhythmic movement is deactivated, the movement is a pure discrete one.

The system would need an extern stimulus to allow the actual system to fit and recover, even when pure discrete movements appear, the phase of the initial rhythmic signal.

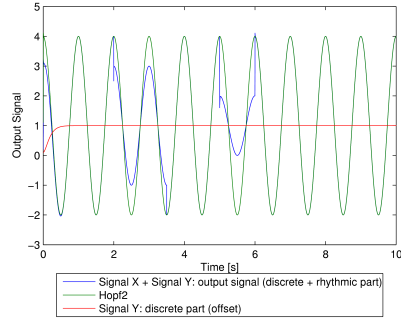


Figure 9: Output signal.(additive / polar coordinates) subject to amplitude changes. Trajectory is modulated by particular choices of $\mu_i > 0$.

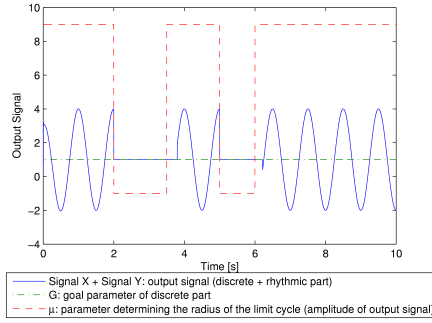


Figure 11: Output signal.(additive / polar coordinates) subject to amplitude changes. Trajectory is modulated by particular choices of μ_i .

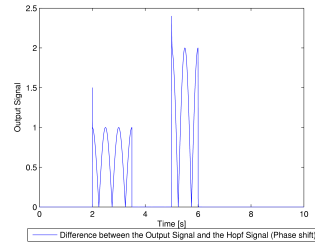


Figure 10: Difference between output signal (additive / polar coordinates) and Hopf oscillator. Trajectory is modulated by particular choices of $\mu_i > 0$.

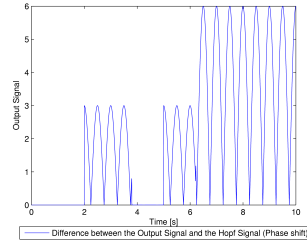


Figure 12: Difference between output signal (additive / polar coordinates) and Hopf oscillator. Trajectory is modulated by particular choices of μ_i .

The external stimulus is already present in the actual implementation. As we can see in figure 3 it is represented by the external clock. It is coupled with the rhythmic component of one CPG.

3 Tempo detection algorithm

Tempo detection and beat tracking consist respectively to measure the tempo and estimate beat's locations in musical signals. Online beat tracking refers to situations where the tempo at a given instant is estimated with access to a limited portion of the future at this time. In offline beat tracking, the whole track is available for processing, which makes the task somewhat easier.

A beat is a pulse on the beat level, the metric level at which pulses are heard as the basic unit. Thus a beat is the basic time unit of a piece; the onset of the corresponding time unit, a point in time, the very moment when the metronome ticks.

Much music is characterised by a sequence of stressed and unstressed beats (often called “strong” and “weak”) organized into a meter and partially indicated by a time signature, the speed of which is determined by a tempo.

The tempo is characterized by the average speed of beats among the music.

As I had no background in signal and music processing, most of the work was to search on the Internet to find theory, examples and library traducing the goal we chose. Two main different approaches arose and were studied deeper. Two can seem little, but firstly the problem of tempo detection is very hard and complicated problem, lot of publications were subject to poor results. Secondly, these two approaches have implementation or library available, which result in a gain of time and especially in a more robust implementation for the code is written by specialist and has already been long used and tested.

3.1 Scheirer Method

In [3], Eric D. Scheirer proposes a method for bmp (beat per minute) detection, also called tempo detection. Four students from Rice University managed to implement the Scheirer algorithm in Matlab. The method acts in 4 steps², figure 13:

3.1.1 Step 1: Filterbank

The signal is divided up into six separate signals, each consisting of the frequency content of the original signal from a certain range. This has the general effect of separating notes from different instrument groups and allowing them to be analyzed separately. Tempo-analyzing the original signal could be error-prone due to conflicting downbeats of different instruments.

²The description of the different steps is inspired by http://www.owl.net.rice.edu/elec301/Projects01/beat_sync/beatalgo.html. More details can be found at this address.

3.1.2 Step 2: Smoothing

Since we are only looking for the tempo of our signal, we need to reduce it to a form where we can see sudden changes in sound. This is done by reducing the signal down to its envelope, which can be thought of as the overall trend in sound amplitude, not the frequencies it carries.

3.1.3 Step 3: Diff-Rect

Now that we have the signals in an enveloped form, we can simply differentiate them to accentuate when the sound amplitude changes. The largest changes should correspond to beats since the beat is just a periodic emphasis of sound.

3.1.4 Step 4: Comb Filter

This is the most computationally intensive step. We need to convolve the differentiated frequency-banded signals with various comb filters to determine which yields the highest energy. A comb filter is basically a series of impulses that occur periodically, at the tempo you specify. Convolution of a comb filter with a total of three impulses with our signal should give an output that has a higher energy when the tempo of the comb filter is close to a multiple of that of the song.

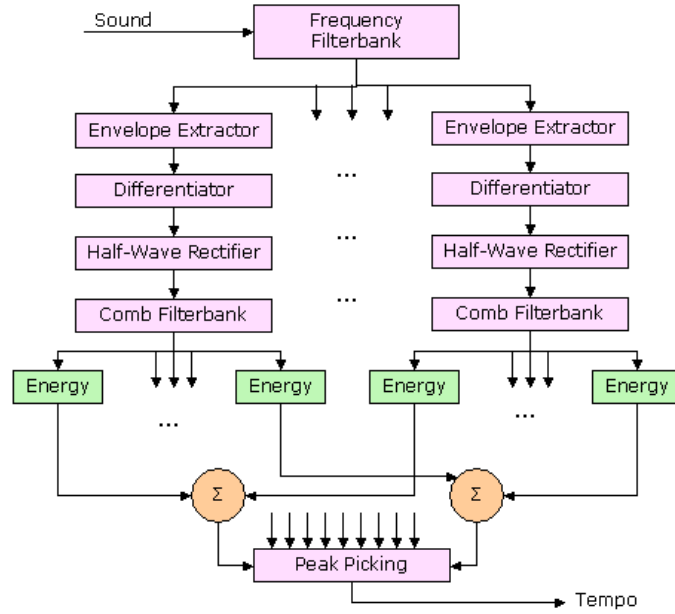


Figure 13: The Scheirer's algorithm's schema.

3.1.5 Matlab implementation

As described, an implementation can already be found on the Internet site. I tried the program with different music instances. Most of the results were good, but as the implementation was only available on Matlab, the quantity of work was important and no library were available for some complex specific function. Regarding the previous reasons, I then tried the second tempo detection approach.

3.1.6 Further work

The first idea about tempo detection was to use the ability of Dynamic Hebbian Learning to reduce a signal to its envelope [5]. This method would have been used for the second step, “smoothing”. Some simulations have been made on this approach (appendix B), but we saw quickly that the time needed to develop such an application was too important for this project. This could be very interesting future research.

3.2 Aubio C Library

The Aubio Library [3] provides automatic labeling features to other audio softwares. Functions can be used offline in sound editors and software samplers. The following functions are implemented in the library:

- various onset detection functions and real time peak-picking
- various pitch detection functions
- beat tracking algorithm
- transient and steady state separation

Aubio depends on the following C libraries : libsndfile, libsamplerate and FFTW. Three linux library providing respectively sound file handling, sample rate conversion in audio and discrete Fourier transform (DFT) implementation.

3.2.1 Beat tracking algorithm³

The beat tracking algorithm used in Aubio is based on [1]. As depicted in figure 14, the procedure involves two stages. The first one is the onset detection and the second one the tempo analysis. In Aubio library, only the first stage which consists in onset analysis is implemented. This allows beat tracking from a given sound file.

³This section is inspired by [1].

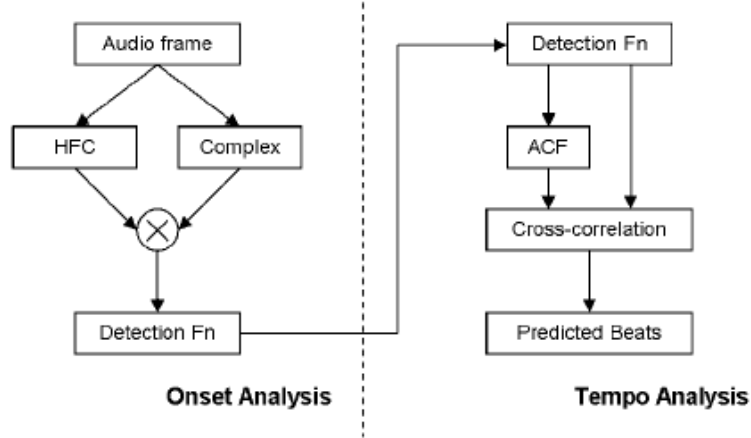


Figure 14: The Algorithm Operation Overview. (HFC stands for high frequency content and ACF for autocorrelation function). Schema reproduced from [1].

3.2.2 Onset Detection

The aim of the onset analysis stage is not to explicitly detect the locations of note onsets, rather to generate a midlevel representation of the input which emphasizes the onset. To reflect this need Davies et al. choose the onset detection function - a continuous signal with peaks at onset positions, as the input to the tempo analysis stage. An example detection function is shown in Figure 15. In their system they use two detection functions - an HFC and a complex domain approach which are multiplied together to create a single input for the tempo analysis stage (as shown in the left hand plot of figure 14). The combination of these two detection functions has been shown to give improved performance for onset detection than when used individually.

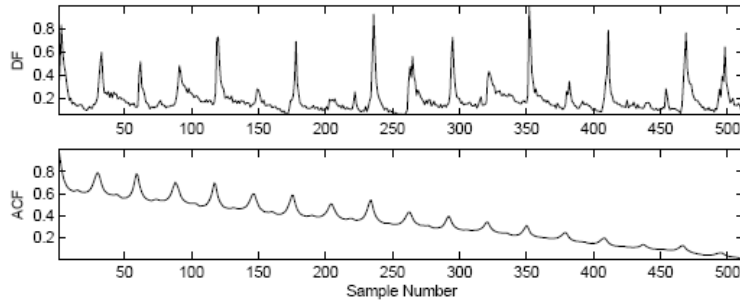


Figure 15: Detection function (upper plot) and corresponding ACF (lower plot).

3.2.3 High Frequency Content (HFC)

Davies et al. use an approach to energy based onset detection, using a linear weighting corresponding to bin frequency k of the Short Time Fourier Transform (STFT) frame $X_k[n]$ of audio input $x[n]$ to emphasize the high frequency energy within the signal, giving the detection function output $df_h[n]$ given by:

$$df_h[n] = \sum_{k=0}^N k |X_k[n]| \quad (9)$$

This technique is particularly appropriate for emphasizing percussive type onsets, most notably cymbals, where the transient region of the instrument hit is mainly composed of energy at high frequencies.

3.2.4 Complex Domain Onset Detection

While the HFC approach is suited for signals with strong percussive content, it performs poorly when applied music with non-percussive onsets, such as those created by a bowed violin. Davies et al therefore incorporate a second detection function that is able to contend with a wider variety of signal types.

The complex detection function $df_c[n]$ shown in equation (10) is a combined energy and phase based approach to onset detection. It portrays the complex spectral difference between the current frame $X_k[n]$ of a STFT and a predicted target frame, $\tilde{X}_k[n]$. Detection function peaks are the result either of energy change or deviations in phase acceleration. These deviations occur in transients as well as during pitch changes (often called tonal onsets, where no perceptual energy change is observed) enabling the approach to detect onsets in a wider range of signals.

$$df_c[n] = \frac{1}{N} \sum_{k=0}^N \|\tilde{X}_k[n] - X_k[n]\|^2 \quad (10)$$

3.2.5 Tempo Analysis

The tempo analysis process is shown in the right of figure 14. Beat period is estimated from the autocorrelation function (ACF) of the detection function.

The tempo analysis stage comes with beat alignment and prediction steps, for improving beat detection. As described by Davies et al, the results are very good and fit the use of a beat detection algorithm for tempo detection.

One thing is, when using offline processing, we use the whole track in order to find the tempo. This results, when tempo inside the music is not constant, to averaging it and result in poor approximations.

3.3 Beat tracking implementation

The Aubio library by Paul Brossier is thus used for implementation. This library offers robust implementation of the Beattracking algorithm. It uses the libsndfile

library in order to open and extract audio content from wav files.

The Mirex Conference⁴ is annual conference which goal is to offer to researcher who work in the domain the possibility to test and compare their approach. They submit practice data along with exact beats list. We used this list with the results of the Aubio implementation to test the efficiency of the method. The results is depicted in figure 16. The method used to compare is the following. For each beat from the solution, we test the beat given by Aubio. If no beat is found, we get 100% error. Otherwise the error is computed by: $\frac{|time_{solutionbeat} - time_{foundbeat}|}{time_{solutionbeat}} \cdot 100$. The results validate the one obtain in [3].

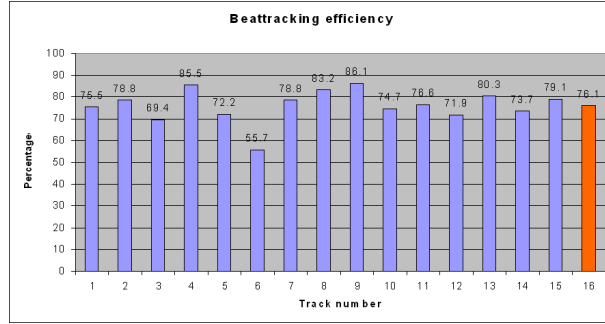


Figure 16: Comparison between given beat list and solution provided by Aubio beattracking algorithm. The average is in orange, 16th column.

The beat tracking algorithm is used for tempo detection. A new library named “Tempo” is implemented. The signature of the main function of this new library is “float tempo(char* filename)”. The function simply receives the name of the audio file to process and return the tempo in beat per minute (bpm).

4 Webots Implementation

Webots is a mobile robotics simulation software that allows fast modeling, programming and 3-dimensional (3D) simulation with physics. More information, API and reference manual can be found on its website⁵.

The simulation is composed of the two main following parts:

- the Webots world: it is the description of the 3D scene. It is described using VRML (Virtual Reality Modeling Language), which is a standard file format for representing 3D interactive vector graphics. A tree representation of the VRML file can be seen in the “scene tree”, directly accessible from the Webots interface (figure 17).
- the controllers: coded in C/C++. They control the different entities present in the world.

⁴http://www.music-ir.org/mirex2006/index.php/Main_Page

⁵<http://www.cyberbotics.com/>

- the physic shared library. It allows to specify personalized behavior to the physic engine (the physic engine ODE is presented in the section 4.2).

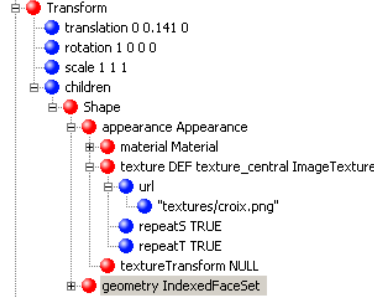


Figure 17: Image of the Webots scene tree. The Transform node displayed is the one used for texture modification on the instruments.

4.1 Original implementation of hoap-2 performing a drumming task

Original implementation (figure 18) on the Webots software was made by Sarah Degallier. The system is composed of the following files:

- drum.c, main controller of hoap-2 robot
- drummer.cpp, C++ drummer class. It inherits DynamicalSystem class from LANDS library.

Two libraries are needed for numerical computation:

- LANDS⁶ is a lightweight object oriented framework for the efficient integration of Nonlinear Dynamical Systems written in C++.
- The GNU Scientific Library (GSL)⁷ is a numerical library for C and C++ programmers.

As the robot performs the drumming task, we need to detect the time at which the sticks hit the instruments in order to generate the music corresponding to the simulation and modify the texture of the instrument for debugging purposes.

Each time a detection is caught, the time and velocity of the contact is written in a file.

⁶<http://birg.epfl.ch/page56667.html>

⁷<http://www.gnu.org/software/gsl/>

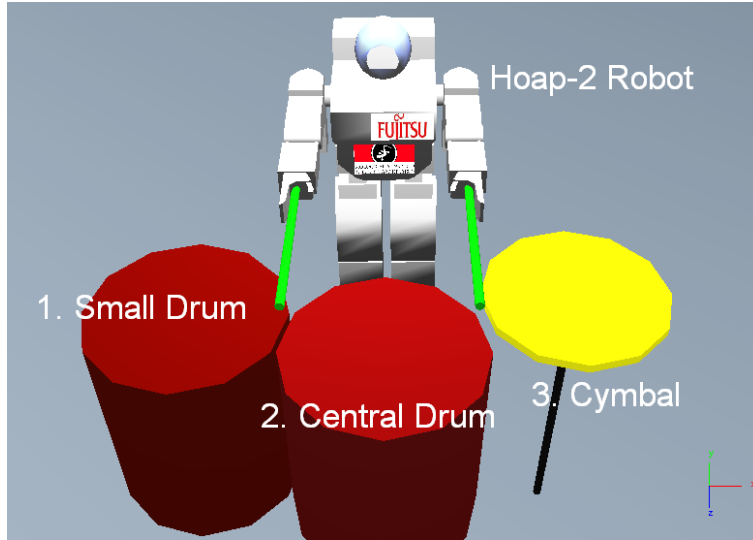


Figure 18: Original Webots world by Sarah Degallier.

4.2 Implementation of Webots collision detection

In order to handle detection, and to offer the possibility to the user to customize the collision reaction and properties, Webots uses the open-source physics engine Open Dynamics Engine (ODE). References and information about this library can be found on its website (<http://www.ode.org>). For the detection of collision between two entities, Webots uses the bounding object node. This object is represented in the world as a simple volume, often a cube or a cylinder, and is used to simplify the interaction with other entities. If the bounding object of two different entities intersects, Webots calls ODE physics to handle the collision. By default, without specifying a personalized physic plugin, Webots manages the collisions on its own by using ODE to perform realistic collision.

4.2.1 The physic plugin

In our case, we must write a specific physic file in order to accomplish the tasks needed when collision arise. In case of collision, we want to keep different information; the time the collision occurred, the instrument hit in the collision and the arm involved. The time is given in second, and corresponds to the time spent in simulation. The instrument is given by number from 1 to 3 specifying the instrument (figure 18):

1. the small drum,
2. the central drum,
3. the cymbal.

The physic plugin file is an interface between ODE and Webots. Adding a custom physics is achieved by creating a custom shared library which is loaded by Webots at run-time and which contains function calls to the ODE physics library.

The WorldInfo node of the simulated world has a field called physics which defines the name of the shared library to be used for the custom physics simulation in the world. This refers to a shared library stored in a subdirectory of the Webots project plugins/physics directory. For our project it is:

WorldInfo {physics : "hand_of_god"}

4.2.2 ODE interface library functions

The shared library⁸ offered six functions that will be called directly by Webots during the simulation of the world. For our purpose, we use three of them:

- void Webots_physics_init(dWorldID w, dSpaceID s, dJointGroupID j);
This function is called upon initialization of the world. It provides the shared library with ODE variables used by the simulation, such as a pointer to the world (dWorldID), a pointer to the geometry space (dSpaceID) and a pointer to the contact joint group used by the simulation (dJointGroupID). Moreover, this function is a good place to call the dWebots-GetGeomFromDEF function to get pointers to the objects on which we want to control the physics.

We use the following links to Webots objects:

- left_hand_geom = dWebotsGetGeomFromDEF("left_hand.1.0");
- right_hand_geom = dWebotsGetGeomFromDEF("right_hand.1.0");
- cymbal_geom = dWebotsGetGeomFromDEF("Cymbal");
- small_drum_geom = dWebotsGetGeomFromDEF("Small_Drum");
- central_drum_geom = dWebotsGetGeomFromDEF("Central_Drum");
- white_ground_geom = dWebotsGetGeomFromDEF("WHITE_GROUND");

The first two geoms relate to the bounding objects of the left and right hand sticks. The three following are the instruments and the last one the ground.

These links to bounding objects will be used in Webots_physics_collide to test if the objects we are interested in are concerned by the collision.

- void Webots_physics_step();
This function is called before every physics simulation step (call to the ODE dWorldStep() function). It has no parameter. It can be used to add force and/or torques to solids. It can also be used to test the position

⁸<http://www.cyberbotics.com/cdrom/common/doc/Webots/guide/chapter6.html>

and orientation of solids (and possibly apply different forces according the position and orientation). This is the function where we analyse the results coming from the following one. It is used to send messages to Supervisor concerning detected changes in collision states.

- `int Webots_physics_collide(dGeomID g1, dGeomID g2);`

This function is called whenever a collision occurs between two objects. It may be called several times for a single simulation step with different parameters corresponding to different objects. We test whether the two colliding objects passed as arguments correspond to the objects we want to control (instruments and sticks). As we just want to recover some information, we let Webots handle the collision with default parameters by returning 0.

The algorithm is quite simple. The procedure for left and right arm is the same, with only variables' name changing. Left arm will be explained here. The order of call of the functions is depicted on figure 19.

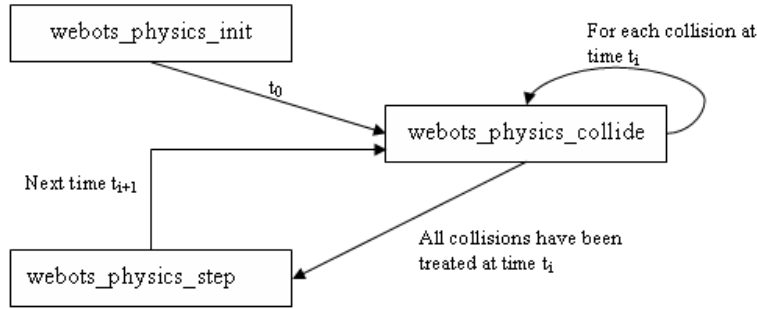


Figure 19: Order of function's call in the physic plugin.

We have two global variables; *left_state* and *left_collide*. *left_state* represents the state of the left arm at time t_{i-1} , *left_collide* represents its state at time t_i . They take for value the number of the instrument the arm is in contact with, or 0 if there is no contact involving the left arm. Both are first initialized to 0 in the init function. Then, in each call to the collide function, we check whether the left arm is in collision with an instrument. If that's the case, the *left_collide* variable is set to the number corresponding to the instrument involved. When the step function is called, we check the *left_collide* variable. Four cases may happen:

1. $left_state = 0$ & $left_collide = 0$. No collision in progress and no new collision this step: no modification and no message sent to the supervisor.
2. $left_state = 0$ & $left_collide \neq 0$. No collision in progress but we encounter a new collision this step: $left_state \leftarrow left_collide$ (instrument number) and we send a message to the supervisor.

3. $left_state \neq 0 \ \& \ left_collide = 0$. Collision in progress but end of the collision this step: $left_state \leftarrow collide(0)$, and we send a message to the supervisor.
4. $left_state \neq 0 \ \& \ left_collide \neq 0$. Collision in progress and continuing this step: : no modification and no message sent to the supervisor.

For the cases 3 and 4, we have a state change; new collision for 2, end of existing collision for 3. We inform the supervisor of that change by sending it a message via the radio emitter/receiver embedded in Webots. The message structure and usage is detailed in section 4.2.4.

4.2.3 Integration of tempo detection

As depicted in section 3.2, Tempo detection is performed by using the beat tracking capacities of the Aubio library. The integration in the drum controller is thus very quick and easy. One can simply use the tempo function to compute offline tempo measurement.

In order to be able to influence on the frequency of the oscillators, which is controlled by the ω parameter, one instance variable called “omega” has been added to the drummer class together with the usual getter/setter function to access and modify it.

Modification to robot’s CPG intrinsic frequency according to a given music’s tempo can thus be implied by the two following steps:

- First, call the tempo function with the name of sound file as argument.
- Secondly, call the setOmega function with the tempo value divided by 60. This traduces the conversion from bpm to Hz.

After each computation of the tempo and modification of the frequency, the new value is sent to the supervisor for on screen information display. The message structure and usage is detailed in section 4.2.4.

4.2.4 Message

The use of messages is mandatory due to the three following reasons:

- when collecting informations about collision, we need to know the actual simulation time. This information is returned by the `robot.get_time()` which cannot be called from the physic plugin,
- the modification of the texture as a visual feedback for collision uses the `supervisor.field.set` function, which as its name says, is only available to the supervisor,
- the display of tempo information on the screen uses the `supervisor.set_label()` function, which is only available to the supervisor.

The first type of message is the one produced by collision detection in the physic plugin. The representation uses an array of float. The message is sent using the emitter/receiver radio embedded in Webots. The sending of a message may have two causes :

- A new collision occurs in the present step.
- A collision finishes in the present step.

The message is sent by the physic plugin and is received by the supervisor, figure 20. The structure of the message can be seen on figure 21.

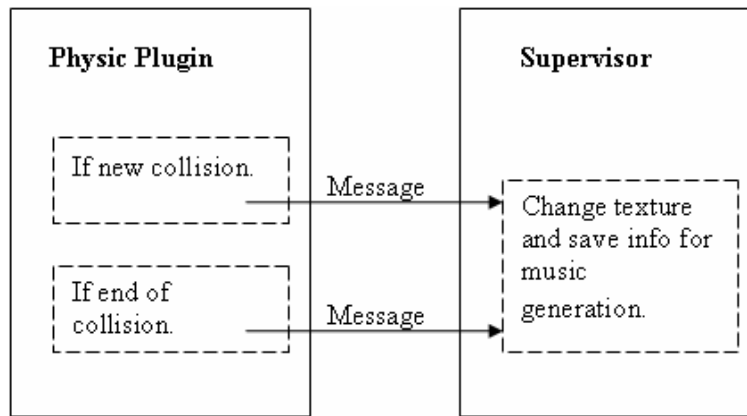
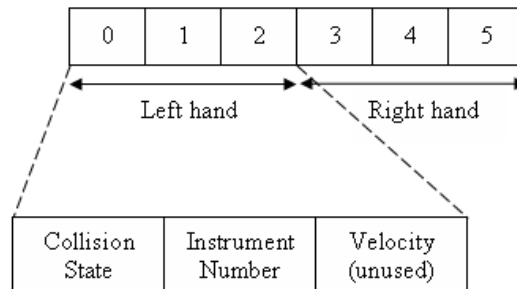


Figure 20: Structure of a message. The different possible values are present on the picture.

Message structure



Values:

Collision State: 0 no change, 1 new collision, 2 end of collision

Instrument Number: 0 no change, 1 small drum, 2 central drum, 3 cymbal

Velocity (default): 1

Figure 21: Structure of a message. The different possible values are present on the picture.

The second type of message is the tempo one. The tempo message is produced by the drum controller after the use of tempo function and setOmega function. This is used to send the new tempo value to the supervisor.

The structure is quite simple. The message is only composed of two numerical values. The first one specifies the number of the file sound used, and the second one the value of the corresponding tempo.

4.2.5 The Supervisor

The supervisor controller is a particular case of a robot controller. A supervisor is a program which controls a world and its robots. For convenience it is represented as a robot without any wheels, driven by a controller with extended capabilities which supervises the whole world.

When the supervisor receives a message from the physic plugin, according to the message information, the texture on the top of the instrument is changed and information for the music generation is stored in a file.

The procedure for the left part of the message (left arm) is explained below; the right part is parsed the same way.

The first number [0] tells the supervisor the behavior it must handle. If it faces to a new collision (1), it changes the “no hit texture”, figure 22, of the instrument involved (given by the second number [1]) by the “hit texture”, figure 23. The supervisor then adds a new line to the file “left.txt” with the time the collision occurred, the instrument involved and the velocity (not used).

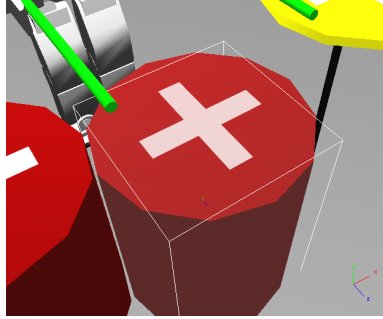


Figure 22: Representation of an instrument in Webots world. The present figure represents an instrument without collision.

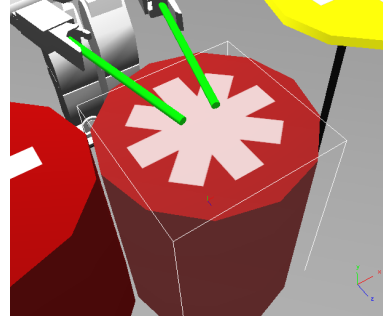


Figure 23: Representation of an instrument in Webots world. The present figure represents an instrument with collision.

When the supervisor receives a message from the drum controller, it uses the supervisor_set_label function to display the new tempo’s value.

4.2.6 Video generation

The video generation is done directly with the embedded “Make movie” option in Webots. The output file extension is “mpeg”. As Webots do not support sound generation, this video comes without any music information.

5 Music generation with Matlab

Matlab has the capacity to deal with wav sound files. The main function involved in this process are `wavread` and `wavwrite`. More information about these functions can be found online⁹.

5.1 Data from Webots simulation

From the simulation, we get two text files. The “right.txt” and “left.txt” files contain for right, respectively left stick the succession of instruments hit, the corresponding time and the velocity.

5.2 Matlab Script

The music generation is done through a self-made Matlab program. The signature of the program is the following:

- `function[] = generation2(Left, Right, Sound1, Sound2, Sound3)`

The function `generation2` takes five arguments. The first two are the matrix produced by the Supervisor (written during the simulation in the “left.txt” and “right.txt” files), containing the information about the simulation’s music. The following three are the songs used to represent, respectively, the cymbal, the central drum and the small drum. The function first creates an empty stereo song, a vector with all zeros at the good frequency. It then copies the sounds into the created song according to the matrix (time and instrument). Two music channels (stereo) are used to translate the spacial position of the instruments.

The resulting sound takes the form of wav sound file.

6 Merging Video and Music

As we now have both video part and sound part from the simulation, we need to find a way to merge these two files. Many free programs offer this feature. VirtualDub on Windows is chosen for the following processing. However, the ideas and technics used here can easily be ported on most of the other free program.

6.1 VirtualDub

VirtualDub is an open source video capture and linear processing tool for Microsoft Windows. It is written by Avery Lee, and is licensed under the GPL. It is hosted on SourceForge¹⁰. It has the advantage to be free, powerful and offer the possibility to control the processing steps with command line. This feature allows to write script and thus automate the merging procedure.

⁹<http://www.mathworks.com>

¹⁰<http://virtualdub.sourceforge.net/>

As two codec, one for video compression and the other for audio compression are used, both are needed to be able to use the script. The video codec is “Xvid Codec”¹¹, and the audio codec is “Lame ACM Mp3 Codec”¹².

Installing “Xvid Codec” is straightforward. To install “Lame ACM Mp3 Codec” you need to “right click” on “LameACM.inf” and select “Install”.

6.1.1 VirtualDub Scripting

The commands’ list is available online¹³. One point to help using such commands, is that we can apply the different compressions, filters and modifications through the graphical interface of VirtualDub, and then save the list of corresponding commands with the “File→Save Processing Settings...”.

6.1.2 Script Generator

The script generator is written in C and offers the completely automated creation of VirtualDub scripts. It works the following way. Firstly, you have to move in the same directory as the program all your movies and sound files. To each sound file corresponds one movie file. Except the extension, both files must have the same name. Example, “testSimulation.mpeg” and “testSimulation.wav” is correct. The program searches all “.wav” files in present in the current folder. It then for each pair of files add the needed command line to the script file called “merging.vcf”.

The processing steps traduced by the VirtualDub commands are the following:

- Open the video file (absolute path).
- Open the audio file (absolute path).
- Set video compression to Xvid.
- Set audio compression to Mp3.
- Merge video and audio and export to new video file.

In our example, the output file would be named “testSimulation.avi”. To use the generated script from command line, the script must be copied to the VirtualDub directory. Then “vdub.exe” must be used with the “/s merging.vcf” option.

Another way is to add the VirtualDub directory to the windows “PATH environment”, and then use a batch file to launch “vdub.exe” from the current folder with “vdub.exe /s merging.vcf”.

¹¹<http://www.koepe.org/xvid.shtml>

¹²http://www.free-codecs.com/LAME_ACM_codec_download.htm

¹³<http://www.virtualdub.org/docs/vdscript.txt>

7 Conclusion

As we can see in the different video available, the objectives of the project are fully reached. The visual feedback in simulation is implemented using the physic plugin. The music generation and merging with the video has also been implemented with success. The tempo processing part of the project is currently offline one. It works fine with constant tempo music. Further implementation should use the capabilities of Aubio for online Beattracking by using the JACK Audio Connection Kit.

Due to the limited knowledge in signal processing, a big part was to find references and literature about tempo detection and music processing. Some bugs were found in the Webots world model and in the controller implementation, they are now corrected.

This project allowed me to discover the signal processing and tempo detection, which is very interesting.

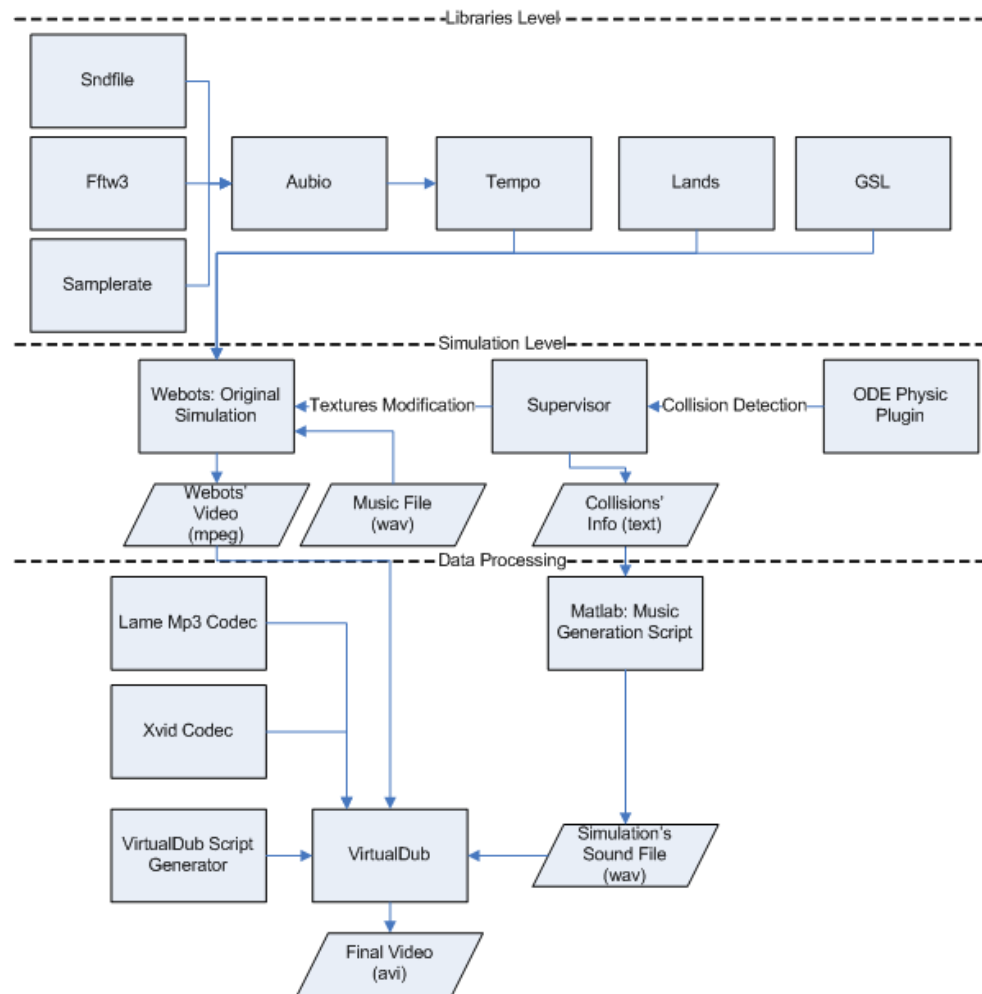
8 Acknowledgment

I thank especially Sarah Degallier for her help. I want to thank the following persons too: Yvan Bourquin for his help with Webots, and Auke Ijspeert for allowing me to perform this project in the BIRG laboratory.

References

- [1] Matthew E. P. Davies and Mark D. Plumbley. Causal tempo tracking of audio. In *In Proceedings of the International Symposium on Music Information Retrieval (ISMIR)*, 2004.
- [2] S. Degallier, C. P. Santos, L. Righetti, and A. Ijspeert. Movement generation using dynamical systems: a humanoid robot performing a drumming task. In *IEEE-RAS International Conference on Humanoid Robots (HUMANOIDS06)*, 2006.
- [3] Paul Brossier Matthew E. P. Davies and Mark D. Plumbley. Beat tracking towards automatic musical accompaniment. In *In Proceedings of the Audio Engeneering Society 118th Convention, Barcelona, Spain*, 2005.
- [4] Olivier Michel. Cyberbotics ltd - webotstm: Professional mobile robot simulation. *International Journal of Advanced Robotic Systems*, 1:39–42, 2004.
- [5] L. Righetti, J. Buchli, and A.J. Ijspeert. From dynamic hebbian learning for oscillators to adaptive central pattern generators. In *Proceedings of 3rd International Symposium on Adaptive Motion in Animals and Machines – AMAM 2005*. Verlag ISLE, Ilmenau, 2005. Full paper on CD.

A Processing Schema



B Dynamic Hebbian learning in adaptive frequency oscillators

As depicted in [5], it is possible to implement oscillators which are able to adapt their frequencies to synchronize with external input. In our case, it will be useful in order to force the oscillator to keep the same rhythm after discrete and rhythmic changes and thus avoid phase shift.

B.1 Mathematical Aspects

Righetti et al. add plasticity to the system, in the sense that the system can change its own parameters in order to learn the frequencies of the periodic input signals. The oscillator can adapt its frequency to any periodic input. The process is completely dynamic, in other words, no offline process is needed. As polar and Cartesian coordinates yield the same results, polar coordinates will be used for these studies.

Given F , the periodic signal with which the oscillator is coupled, the coupled Hopf oscillator is governed by the following differential equations:

$$\dot{r}_i = (\mu_i - r_i^2) r_i + \epsilon F \cos \phi \quad (11)$$

$$\dot{\phi}_i = \omega - \frac{\epsilon}{r_i} F \sin \phi \quad (12)$$

$$\dot{\omega}_i = -\epsilon F \sin \phi. \quad (13)$$

Without perturbation, when $\epsilon = 0$, the system is oscillating at $\omega \text{ rad s}^{-1}$. This oscillator is coupled with a periodic force F . When the force is zero, the system has an asymptotically stable harmonic limit cycle, with radius $\sqrt{\mu_i}$ and frequency ω . Equation 13 is the learning rule for the ω parameter in eq. 12. ω will converge to such value that one frequency component of the oscillator and one of the input F match. The adaptation of ω happens on a slower time scale than the evolution of the rest of the system. This adaptation time scale is influenced by the choice of ϵ .

B.2 Time needed for synchronization with coupled oscillator's frequency

Figure 24 shows the time needed for synchronization for different frequency distance (difference between the original frequency and the goal frequency). We can see that the time does not increase linearly but exponentially.

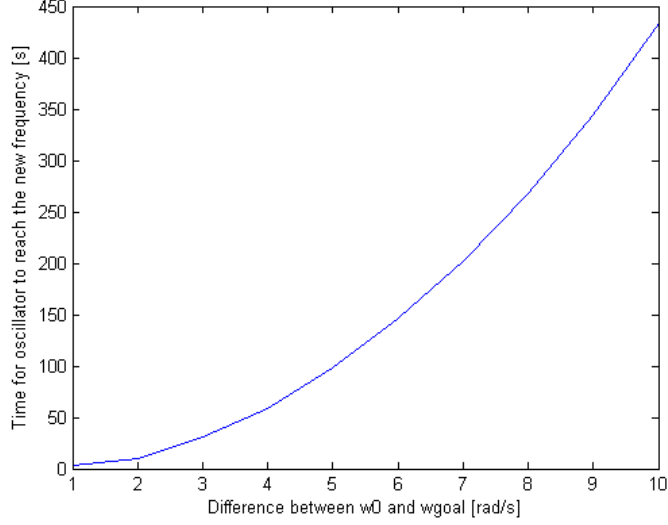


Figure 24: Time needed for Oscillator using Hebbian Learning rule for synchronization with external force.

B.3 Applying the Dynamic Hebbian learning in adaptive frequency oscillators to the dynamical system of the CPG

The Dynamic Hebbian learning has been applied to the dynamical system capable of both discrete and rhythmic signal (section: 2.2). It is coupled with a Hopf oscillator with parameter μ_i (amplitude) equal to 1 and g_i (offset) to 1. The results are shown in figure 29.

The main point is that the new added oscillator's capacity makes perfectly their job. The stability of the system achieves good results. The error, despite the important changes in terms of amplitude and discrete movements, returns to the initial value at the end of the experiment, when the parameters are set back to initial values (same parameters as Hopf oscillator). The small phase shift, present at the beginning as well as at the end, is due to the Dynamic Hebbian system itself. Figures 25, 26, 27, 28, 29, 30.

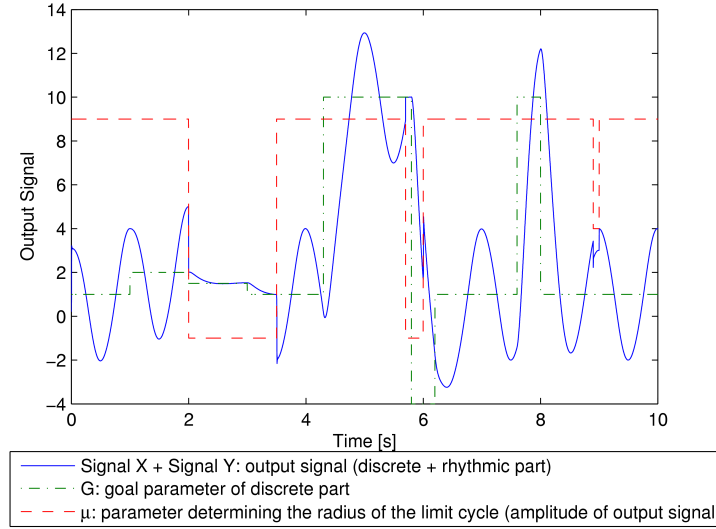


Figure 25: Progression of output signal (Additive / Cartesian coordinates / Dynamic Hebbian learning) subject to discrete and rhythmic movements with Hopf oscillator. Trajectory is modulated by particular choices of μ_i and g .

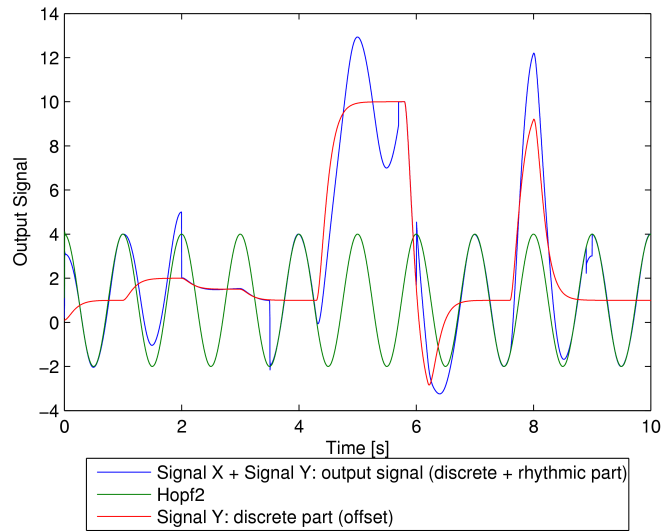


Figure 26: Progression of output signal (Additive / Cartesian coordinates / Dynamic Hebbian learning) subject to discrete and rhythmic movements. Trajectory is modulated by particular choices of μ_i and g .

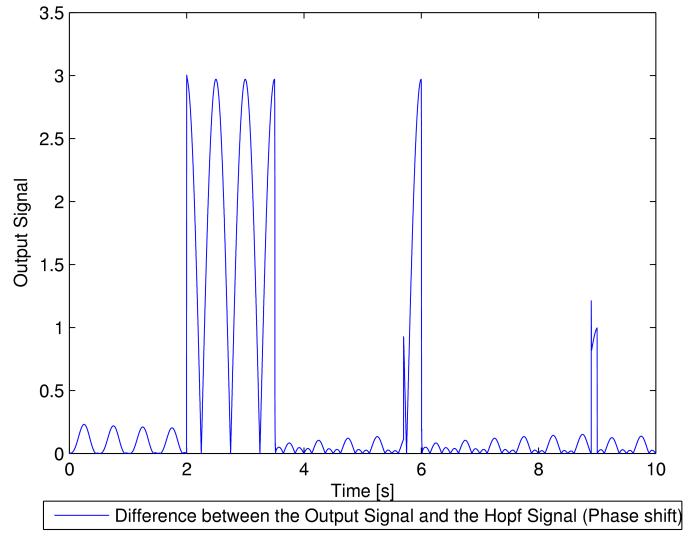


Figure 27: Difference between output signal.(Additive / Cartesian coordinates / Dynamic Hebbian learning) and Hopf oscillator.

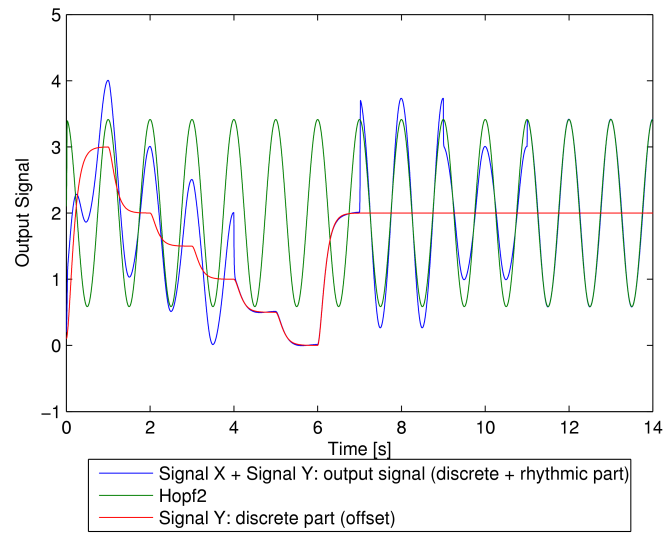


Figure 28: of output signal (Additive / Cartesian coordinates / Dynamic Hebbian learning) subject to discrete and rhythmic movements with Hopf oscillator. Trajectory is modulated by particular choices of μ_i and g .

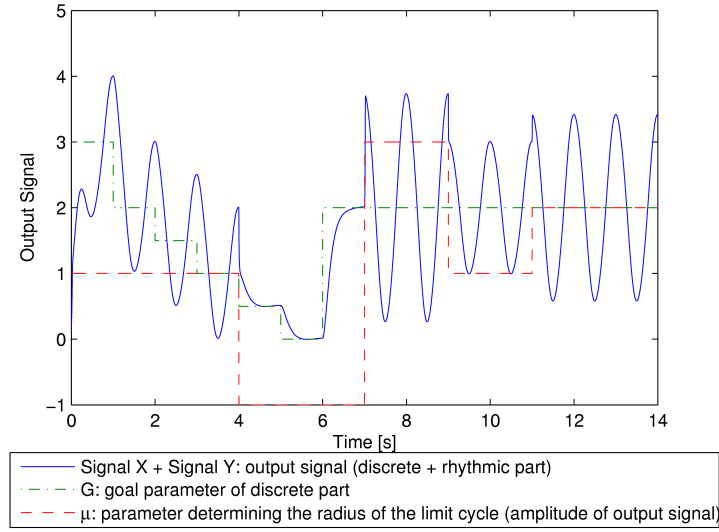


Figure 29: of output signal (Additive / Cartesian coordinates / Dynamic Hebbian learning) subject to discrete and rhythmic movements. Trajectory is modulated by particular choices of μ_i and g .

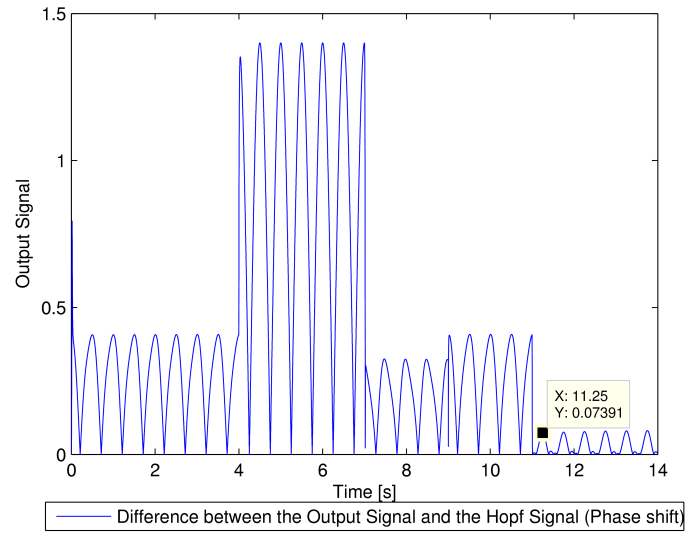


Figure 30: Difference between output signal.(Additive / Cartesian coordinates / Dynamic Hebbian learning) and Hopf oscillator.

C Collision detection issue

I had got some problem with collision detection. Indeed, the physic plugin detected collision when no sticks were hitting the instruments at all. The problem was not so easy to find and came from the model of the hoap-2 robot. The bounding objects corresponding to the stick is misplaced in the original model, figure 31. The bug is corrected in third version of the world.

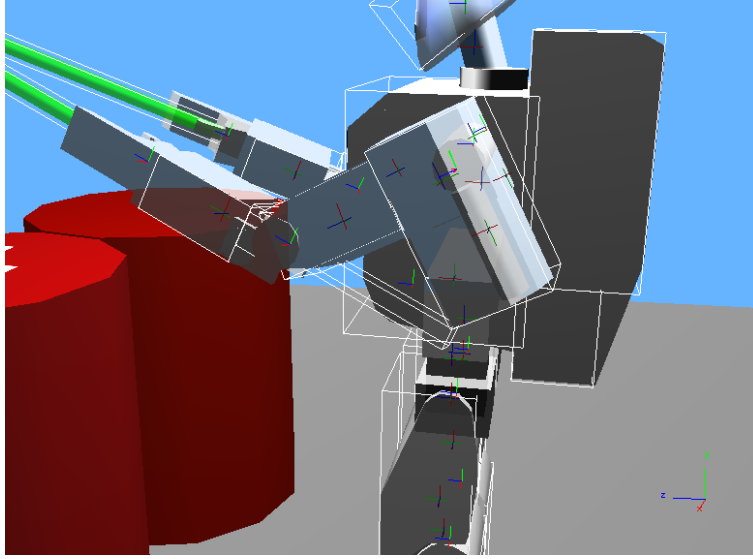


Figure 31: Original bounding object for left and right stick. We can see the exceeding size which causes detection errors.

D Screenshots

