

EXTRACTING MUSICALLY-RELEVANT RHYTHMIC INFORMATION FROM DANCE MOVEMENT BY APPLYING PITCH TRACKING TECHNIQUES TO A VIDEO SIGNAL

Carlos Guedes

Escola Superior de Música e das Artes do
Espectáculo (ESMAE-IPP)
Porto, Portugal

Escola Superior de Artes Aplicadas (ESART-IPCB)
Castelo Branco, Portugal
e-mail: carlosguedes@mac.com

ABSTRACT

In this paper, I discuss in detail the approach taken in the implementation of two external objects in Max/MSP [11] that can extract musically relevant rhythmic information from dance movement as captured by a video camera. These objects perform certain types of analysis on the digitized video stream and can enable dancers to generate musical rhythmic structures and/or to control the musical tempo of an electronically-generated sequence in real time. One of the objects, **m.bandit**,¹ implements an algorithm that does the spectral representation of the frame-differencing video analysis signal and calculates its fundamental frequency in a fashion akin to pitch tracking. The fundamental frequency of the signal as calculated by this object, can be treated as a beat candidate and sent to another object, **m.clock**, an adaptive clock that can adjust the tempo of a musical sequence being played, thereby enabling the dancer the control of musical tempo in real time.

1. INTRODUCTION

m.bandit and **m.clock** are the core objects of a small library I created for interactive dance performance (called the m-objects²) that extract what I call *musical cues* from dance movement in real-time. *Musical cues* are rhythms produced by movement in dance that bear qualities akin to musical rhythms in their durational and articulatory nature. As noted by Fraisse [4] [5], and Parncutt [16], the periodicities found in musical rhythms lie between 200 and 1800 ms. This range is also the one that allows for motor synchronization with a sound stimulus. The *musical cues* extrapolated by the software are thus the articulations in dance movement whose time spans fall within that range.

During an interactive dance performance utilizing the m-objects, a musician can utilize the *musical cues* extracted from dance movement to either generate rhythm to animate a musical sequence or to enable the dancer the control of musical tempo of an electronic music score in real time.

The objects process data from the frame differencing analysis of a digitized video stream. Frame (or image) differencing is a motion segmentation technique ([21], Chapter 11) that subtracts the luminance value of the

pixels between two frames. A common application for frame differencing in video analysis is the detection of the *quantity of motion* (the amount of light change) that occurred between two frames. Frame differencing is available in several objects that process digitized video in the Max/MSP programming environment. For example, Eric Singer's video analysis object **Cyclops** [12] or the object **v.motion**, which belongs to David Rokeby's softVNS 2 image analysis library for Max/MSP [10], can compute the quantity of motion between two consecutive frames in a digitized video stream. The object **m.bandit** outputs the spectrum of the time-domain representation of the frame-differencing video analysis signal and its fundamental. The object **m.clock** is an adaptive clock that takes as input the output from **m.bandit**, and allows changing the tempo of a sequence being played according a very simple system of rules. After describing certain properties of the frame-differencing video analysis signal, I do a detailed description about how the spectral representation of dance movement is implemented as well as how the musical tempo control in real time can be done using these objects.

2. PROPERTIES OF THE FRAME DIFFERENCING ANALYSIS SIGNAL AND ITS REPRESENTATION IN THE TIME DOMAIN

If a camera is looking at a controlled-lighting environment in a fixed position with a static background and we have an object moving against that background, we can detect the *amount of motion* that occurred between consecutive frames. This can be done by calculating the difference in luminance of each pixel of the image. In the areas of the image in which no motion has occurred, the difference is zero, since the pixels of that area carried the same luminance onto the next frame. In the areas in which motion has occurred the absolute value of the difference will be greater than zero.³ Summing all the luminance difference values of each pixel in two consecutive frames corresponds to the overall change that occurred in the pixels that changed

¹ Max/MSP objects appear in bold typeface in the text.

² A succinct full library description can be found in [9].

³ Ideally, the moving object should have a contrasting color against the background (e.g. white object moving against a black background or vice-versa). In general some denoising has to be done. Digital cameras usually introduce some noise in the digitized stream. This can be easily overcome by *thresholding*, i.e., by introducing a threshold value in which the calculated brightness differences are forced to zero when they are below that value [21].

their value, or the amount of motion that occurred between those two frames.

When we plot over time the instantaneous values given by the sum of the absolute value of the differences in luminance of all the pixels in a stream, all the changes detected through frame differencing will correspond proportionally to the amount of movement performed by the moving body. The more the body moves, the greater will be the number of pixels that changed their luminance between consecutive frames. When there is movement inflexion (e.g. a change of direction) the value of the sum in the luminance difference tends to zero since there was a sudden stop in the movement for the inflexion to occur.

If we compute the number of pixels that changed their brightness in periodic movement actions over time, we can detect periodicities in the video analysis signal that are in direct correspondence to the actions performed. This works very well for simple periodic actions such as jumping or waving a hand, for example. Moreover, wider actions correspond to increasing the amplitude of the frame-differencing signal and faster actions correspond to increasing the frequency (see Figure 1).

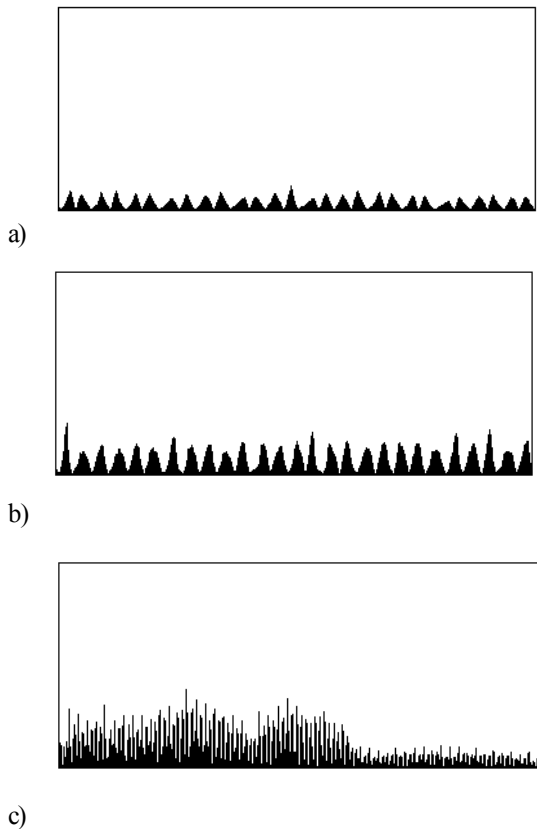


Figure 1. Several representations in the time domain of frame-differencing graphs of movement of waving hand. Figures 1a) and b), same frequency, with two different amplitudes; 1c), faster frequency with amplitude variation.

If we remove the DC offset from the signal, the similarities to periodic acoustic signals are simply striking (see Figure 2). This means that if we apply an algorithm that detects the periodicities of this signal — such as the Fast-Fourier Transform (FFT) — that works

for frequencies below the audible range, we can detect the rhythms present in the signal. Moreover, if we apply an algorithm for pitch detection that works in the same frequency range we can detect the fundamental frequency (or tempo) of a periodic movement action, and thus use this information for doing something musically relevant, such as generating musical rhythms from dance movement or enabling a dancer to control the musical tempo in real time.

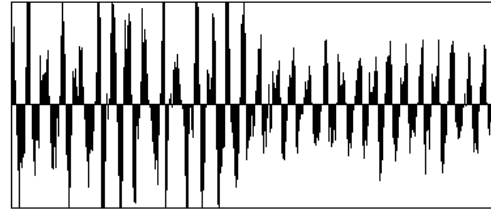


Figure 2. DC offset removal of the quantity of motion variation in a video caption of a waving hand.

3. THE SPECTRUM OF A DANCE: REPRESENTING THE FRAME-DIFFERENCING SIGNAL IN THE FREQUENCY DOMAIN

In order to get a spectral representation of the signal, **m.bandit** employs a bank of 150 second-order recursive band-pass filters. The center frequency of these filters spans from 0.5 Hz to one-half of the sampling/frame rate⁴ is distributed logarithmically,⁵ and their bandwidth is proportional to the center frequency (10% of the center frequency).⁶ The Goertzel algorithm ([6]; see also [14], Chapter 6, pp. 287-289) is applied to the output of the band-pass filters in order to get a magnitude and phase representation of each center frequency.

The main reason for applying a bank of band-pass filters using the Goertzel algorithm, as opposed to FFT, in order to obtain the frequency-domain representation of the signal has to do with the speed of computation at runtime. The Goertzel algorithm can compute the Discrete-Fourier Transform (DFT) of a single frequency component [2][14] faster than FFT [1][20]. The Goertzel algorithm thus allows the calculation of the magnitude and phase spectrum of a selected frequency band, and when using a bank of band-pass filters one can select the frequency range at which the object is operating, also sparing calculations of other unwanted frequency bands

⁴ Ideally, 25 or 30 frames per second (25 or 30 Hz) depending on the broadcasting system one is using (PAL or NTSC).

⁵ This is an important feature to consider for the detection of pulse in dance movement. **m.bandit**, the Max external object that performs the frequency-domain representation of the time-domain representation of the frame differencing signal, outputs a visual representation of the amplitude spectrum of the signal. Having a logarithmic distribution of the frequencies is very useful in order to visualize the relationships between the frequencies that comprise the signal. The prominent harmonics of a fundamental frequency will keep the same relative distance between them in the visual representation. Harmonic relationships between the frequencies comprising the signal are a fundamental aspect for the determination of pulse. This issue will be discussed in detail later in the paper.

⁶ The best value for the bandwidth was determined by trial-and-error, and seems to be the best value for this type of application.

at runtime.⁷

The frame-differencing signal spectrum is calculated in two stages. The first stage consists in passing the time-varying representation of the frame-differencing signal through the bank of band-pass filters. The difference equation for the second-order band-pass filter is:

$$Y_{(n)} = X_{(n)} + 2CR \times Y_{(n-1)} - R^2 \times Y_{(n-2)} \quad (1)$$

$$C = \cos\left(\frac{2\pi cf}{sr}\right)$$

$$R = e^{\left(\frac{-\pi bw}{sr}\right)}$$

cf = center frequency

bw = bandwidth

sr = sampling rate

$e = 2.718281828459045...$ (Euler's number)

This equation can be found in Stan Tempelaars's book [20] as an example of a formant filter for synthetic speech production.

The second stage in the calculation of the spectrum extracts the real and imaginary parts of the output by applying a slight alteration of the Goertzel algorithm as suggested by Peter Pabon [15]. The real and imaginary part of each filter's output are calculated as follows:

$$Y_{real} = Y_{(n)} - R \cos\left(\frac{2\pi bw}{sr}\right) \times Y_{(n-1)} \quad (2)$$

$$Y_{imag} = -R \sin\left(\frac{2\pi bw}{sr}\right) \times Y_{(n-1)} \quad (3)$$

The magnitude and phase components for each frequency band can then be easily calculated:

$$Y_{mag} = \sqrt{Y_{real}^2 + Y_{imag}^2} \quad (4)$$

$$Y_{phase} = \arctan\left(\frac{Y_{real}}{Y_{imag}}\right) \quad (5)$$

In the object **m.bandit**, each band-pass filter is represented as a C programming language structure, where all coefficients are stored upon the object's instantiation (see Figure 3). The Max external object definition contains an array of 150 of such structures for spectrum calculation. The signal magnitude spectrum is output as an Atom data type — a data type used to output lists in Max — and can be visualized by Max's **MultiSlider** object (Figure 4).

```
typedef struct filter //filter structure
{
    double y[3];      //output(y(n)), previous
                      //(y(n-1)),second
                      //previous (y(n-2))

    double f_cf;      // center frequency
    double f_bw;      // bandwidth
    double f_c;       //coefficient C
    double f_r;       //coefficient R
    double f_cr;      //coefficient CR
    double f_two_cr;  //coefficient 2CR
    double f_rsqr;    //coefficient R^2
    double f_out;     //output
    double f_real;    //real part
    double f_imag;    //imaginary part
    double f_rsin;    //RsinPhi
} Filter;
```

Figure 3. C structure representing a digital band-pass filter and all of its pre-calculated coefficients in **m.bandit**.

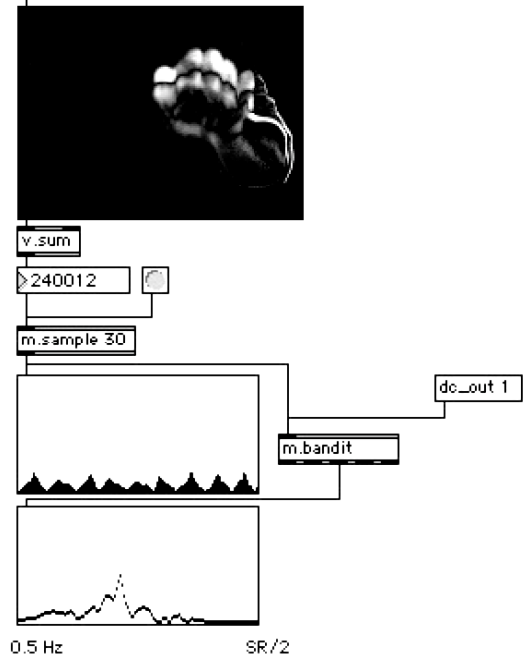


Figure 4. Magnitude spectrum output by **m.bandit**.

4. THE MUSICAL TEMPO OF A DANCE: DETERMINING THE FUNDAMENTAL FREQUENCY OF THE FRAME DIFFERENCING SIGNAL

4.1. Beat Tracking as Pitch Tracking

Determining the pulse of a musical sequence is essentially determining the frequency and phase of a very low frequency [19][18]. The problem of beat tracking can be seen as a special case of pitch tracking in which phase detection plays a more predominant role. As noted by Eric Scheirer [19] human pitch recognition

⁷ A future implementation of **m.bandit** will allow the user to dynamically allocate the number of filters to use, the frequency range and their bandwidth. This will allow to 'tap' selectively a certain frequency range, and/or to use several of these objects without sacrificing the speed of computation.

is only sensitive to phase under certain conditions, whereas “rhythmic response is crucially a phased phenomenon — tapping on the beat is not at all the same as tapping against the beat, or slightly ahead of or behind the beat, even if the frequency of tapping is accurate” (p. 590).

Certain existing computational algorithms of beat tracking attempt to find the tempo of a musical sequence by estimating the frequency and phase of the most prominent pulse in a musical sequence. These algorithms utilize adaptive oscillator models (see for example [13][23]), banks of band-pass and comb filters [19], or adaptive filters [3].

On the adaptive oscillator models approach [13][23], a network of non-linear oscillators take as input a stream of onsets and continuously adapt the period and phase of an oscillator to match the tempo of a musical sequence. Scheirer’s oscillator model [19] is inspired on Large and Kolen’s approach [13] and uses a network of resonators (comb filters) to phase-lock with the beat of the signal and determine the frequency of the pulse. Scheirer’s model operates in acoustic data rather than on event streams, and more pre- and post- processing is required in order to accurately extract the beat in a musical sequence. The adaptive filter model presented by Cemgil et al. [3] formulates tempo tracking in a probabilistic framework in which a tempo tracker is modeled as a stochastic dynamical system. The tempo is modeled as a hidden state variable of the system and estimated by Kalman filtering.

The proposed approach for determining the underlying tempo of a dance sequence is largely inspired on the approaches mentioned above. This approach — which has not been tested on musical data — seems to provide very good results for the work in interactive dance, in terms of determining the most prominent frequencies that can be considered the tempo of a movement sequence. The approach presented here should not therefore be considered as an alternative model for beat tracking in music.

In the previous section, it was shown how one could determine the frame-differencing signal spectrum by applying a modified band-pass filter bank to that time-varying signal. The object **m.bandit** can also output the most prominent instantaneous frequency of that spectrum. This is done by cross-correlating the calculated spectrum at a certain instant with the spectrum of a 1 Hz pulse train. Unlike the beat trackers presented above, **m.bandit** does not calculate the phase of the most prominent frequency. In fact, it ignores phase completely and this makes this algorithm very akin to other algorithms of pitch tracking. The reason for ignoring the phase spectrum of the signal will be explained later in the paper.

4.2. Determining the Musical Tempo of a Dance

The musical tempo detection of a dance performed by the system implements some knowledge about pulse sensation. According to Parncutt [17] pulse sensation is an inherent quality of musical rhythm. It is an acoustic phenomenon produced by the interaction between the low-frequency periodicities present in a musical sequence. Parncutt equates the perception of pulse to the

perception of pitch in complex tones. Pulse sensation is enhanced by the existence of *parent* or *child* pulse sensations that are *consonant* with the most prevalent one.

If we want to determine the *musical* pulse in dance movement one should analyze if the periodicities in the motor articulations of dance movement evoke the sensation of regularity akin to musical pulse. That is, if these articulations bear the durational characteristics of musical rhythms and if they are in a certain proportion to each other in a way that pulse is reinforced. The approach utilized for determining the pulse in a movement sequence is the same that can be applied for finding the fundamental frequency of a pitched complex tone. It is based on a harmonicity criterion, i.e., that the pulse sensation is enhanced by periodicities that are integer multiples of a beat duration as proposed by Parncutt.⁸ Hence, the cross-correlation with a 1 Hz pulse train in order to find the frequency that can be a beat candidate. The spectrum of a 1Hz pulse train contains harmonic peaks, and the output of the spectral cross-correlation will thus favor signals that contain harmonic peaks.

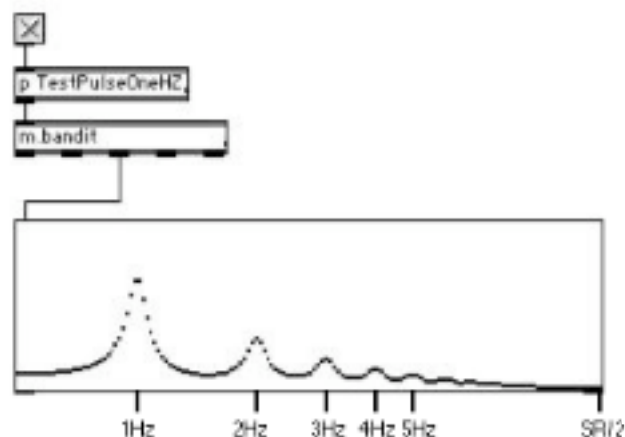


Figure 5. Spectrum of 1 Hz pulse train as calculated by **m.bandit**.

The object **m.bandit** embeds a file called `testsig.h` in which the amplitude spectrum of a 1 Hz pulse train is stored. This spectrum was collected from **m.bandit** itself. The array in which the pulse train spectrum is stored, contains a simplified version of the actual output: the amplitude value of the output for each frequency band was normalized to 1, and all the amplitude outputs for the frequency bands, except those that correspond to the harmonics of the one-hertz pulse, were forced to zero. The array only contains the amplitude values of the frequencies between 1 and 4 Hz (63 points), the first four harmonics of the one-hertz pulse train.

⁸ Parncutt [17] actually says that pulse sensation is enhanced through the existence of parent or child pulse sensations. This means that pulse sensation should also be enhanced by the presence of periodicities that are in sub-harmonic proportion to the pulse. The calculations performed by **m.bandit** only take into account the periodicities that are in harmonic proportion to a given pulse. This simplification does not seem to have an influence on the detection of the correct pulse.

```
#define TEST 63
```

```
float testsig [TEST]={0.95362, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.52969, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0.38007, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0.30061};
```

Figure 6. File testsig.h.

The cross-correlation between the instantaneous spectrum of the signal and the spectrum of the 1 Hz pulse train is done the following way: starting at the lowest frequency band (0.5 Hz), the first 63 points of the spectrum are multiplied by the corresponding values in the testsig array. All the multiplication results are added to each other and the final sum is stored temporarily. This operation is repeated up to the 88th center frequency band (around 3.4 Hz) of the band-pass filter bank. The index containing the highest sum value is the index of the center frequency that has highest correlation with the one-hertz pulse train. That center-frequency value is returned by **m.bandit**'s leftmost outlet (see Figure 7).

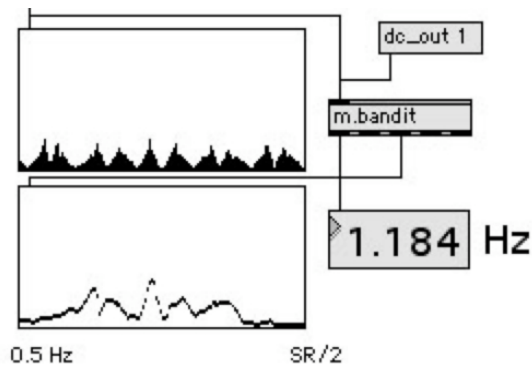


Figure 7. **m.bandit** outputting the instantaneous value of a beat candidate.

m.bandit is not a beat tracker in the true sense of the word. This object is able to do an instantaneous estimate of the fundamental frequency of a low-frequency signal, and outputs values that can be considered musical beats. It implements knowledge that helps providing good estimates about the most prominent beat in a movement sequence but, since it only outputs instantaneous values, it does not compare them or provides in anticipation what the next value will be. It is not even able to analyze if a movement sequence has, in fact, a steady beat or not.

If one wants to do a temporal analysis of the output of **m.bandit** (beat tracking, for example) one should connect the outlet that outputs the fundamental frequency estimate to other existing objects in the library. The object **m.clock** models a simple tempo tracker that takes the fundamental frequency output from **m.bandit** as input, and produces a tempo estimate based on that data. **m.clock** can be effectively used to enable a dancer to control the tempo of a musical sequence in real time.

4.3. Ignoring Phase

As mentioned earlier, **m.bandit** does not calculate the signal phase spectrum and, consequently, the software that was developed is not phase sensitive. This decision was taken after doing some experiments with earlier versions of **m.bandit** that calculated the signal phase spectrum and output a ‘bang’ message whenever the phase zero-cross of the fundamental frequency occurred.

Having a system that is phase sensitive seemed initially a good and logical idea to implement. However, it was gradually abandoned. The main reason has to do with the fact that **m.bandit** is outputting the estimate for the fundamental frequency of the signal at frame rate, and this information tends to vary considerably since there is some noise in the system. The camera grabs the images, the computer digitizes them, and some processing still has to be done in order to provide the frame-differencing signal for analysis. Very small delays in this processing cycle can introduce errors in the measurements, which means that a sudden jump in value in the frequency estimate would cause the phase synchronization to jump as well. A statistical approach to the measurement of the tempo estimate seems to be the appropriate thing to do in this system.

Also, when the dancer accelerated or decelerated the tempo in a movement sequence, the phase zero-cross would jump abruptly for every new fundamental frequency estimate. This implied that a lot of denoising on that information, such as ways of smoothing out these changes in phase, had to be done at runtime. Even so, I still thought that the phase zero-cross message output could be used as a musical rhythm generator, but this information needed to be quantized so that the rhythms created by this generator fell in places according to the beat. The idea of utilizing the phase of the detected fundamental frequency was therefore abandoned.

Furthermore, trying to build a system that is phase sensitive for musical tempo control in dance does not seem to be that crucial after all. In interactive music performance, having a system that can accurately follow a player’s positioning in tempo is a crucial aspect since following the performer asynchronously can jeopardize the performance of the piece. As poignantly pointed by Scheirer [19] the positioning in time of musical events is a crucial aspect in music performance. However, in dance performance (interactive or not), the music tends to provide a temporal grid from which dancers can move in and out, and synchronize to it or not. In fact, this aspect — somehow typical in the temporal interaction between dance and music — can create a powerful audiovisual counterpoint between the rhythmic articulations of both during performance.

The musical tempo control features provided by the library model this latter aspect in dance performance. Since the system is not sensitive to phase, the dancer can control the tempo of a musical sequence asynchronously. However, triggering musical events or sequences from the dancer’s actions is contemplated by the library through the object **m.peak**, that outputs ‘bang’ messages when a peak above a certain threshold is detected, thereby allowing the triggering of musical events or sequences when a dancer performs a rather accentuated body action.

5. CONTROLLING MUSICAL TEMPO FROM DANCE MOVEMENT

In order to enable a dancer the control of musical tempo, the fundamental frequency of **m.bandit** is analyzed by **m.clock** at frame rate. **m.clock** is an adaptive clock that outputs and adapts to the tempo according to certain rules. The best description for **m.clock** behavior is that of a filter. **m.clock** receives from **m.bandit** a tempo estimate, and filters out any unwanted information for musical tempo estimation and adaptation. It could be said that this object acts in part as a denoising filter for **m.bandit**.

The adaptive clock embedded in **m.clock** is modeled according to the following formula:

$$T_{(n)} = T_{(n-1)} + \Delta t \quad (6)^9$$

$T_{(n)}$ is the clock value in milliseconds at frame n , $T_{(n-1)}$ is the clock value at the previous frame, and Δt is the difference between the measurement that was output by the band pass filter bank (converted to milliseconds) and $T_{(n)}$. Figure 8 illustrates how the output from **m.bandit** can be connected to **m.clock**.

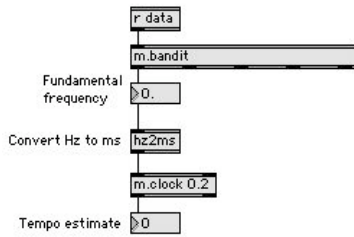


Figure 8. Simple patch illustrating the connection between **m.bandit** and **m.clock**.

This clock only works for values that can be considered musical beats — 300 to 1500 milliseconds or 3.33 to 0.66 Hz [16]. Each time **m.clock** receives a new value, the clock object checks if the value is within the musical beat boundaries. If the value is not within boundaries, that value is ignored and no calculations are performed. When a legal value arrives, if it is the first one to be within boundaries and if no default initial value was sent to **m.object**, $T_{(n)}$ gets initialized to that value.¹⁰ For

⁹ Elsewhere [7], I proposed a slightly different formula for the clock behavior, in which $T_{(n)} = aT_{(n-1)} + b(T_{(n-1)} + \Delta t)$. The coefficients a and b could be used to set the degree of “strictness” of the clock. a and b had to be both greater than zero and $a = 1 - b$. This was intended to allow further resistance to tempo change by the clock. If the user wanted the clock to be extremely strict, coefficient a could be set to a value close to 1. This would make the clock very little sensitive to tempo changes induced by the dancer. This formula worked well with videos that were being used for analysis at the time, in which the dancers were dancing to music with very strict tempos (e.g. samba). However, during the first experiments with the clock operation in real time using live input, I realized the simplification of the initial formula presented above worked better. The main reason has to do with the fact that the user only has to worry about the control of one free parameter for the clock’s calculation (the allowable amount for tempo deviation) as opposed to two (amount of variation and strictness).

¹⁰ **m.clock** can also be initialized with the message ‘accept t’ (t is the a time value in milliseconds between beat boundaries). This is a

subsequent legal beat values, the clock object checks if the variation between the received value and the current clock time is within the allowable margin for variation. If it is, the new tempo estimate is computed according to equation 6. If it is not, the current tempo estimate remains unchanged ($T_{(n)} = T_{(n-1)}$).

5.1. Performing the Tempo Adaptation

The allowable margin for variation (delta) is a percentage of the current beat value, defined by the user. This value, expressed as a floating-point number between zero and 1, is multiplied by the previous tempo estimate (delta * $T_{(n-1)}$) each time a new tempo estimate is calculated. When a new legal beat value is accepted for calculation, **m.clock** calculates the *absolute difference* between that value and the current tempo, and checks if that value is within the allowable margin. If the value is within that margin, Δt becomes the *difference* between the accepted value and $T_{(n-1)}$, and the current tempo estimate is updated. When one wants to control the tempo of a musical sequence through the output of **m.clock** (for example, to control the tempo Max’s **metro** object), one should interpolate between values (for example, using the **line** object) in order to smooth the output. The amount of allowable variation for tempo adaptation can be input as an argument to the object. The amount of variation can also be changed by sending the message ‘delta f’ to the object (f is a floating-point value between zero and 1). If no values are introduced when **m.clock** gets instantiated, the object initializes the allowable margin for change to the default value of 15% of the current beat value (delta = 0.15).

An adaptive clock that exhibits the behavior described above can have many interesting applications for interactive dance performance when applied as an instrument that enables the musical tempo control by a dancer. As mentioned before, this clock only adapts to the new tempo if the new measurement is within an allowable margin for change. This allows, for example, that a dancer moves rhythmically outside of the current tempo and no tempo changes will occur. If the dancer wants to change the tempo he has to synchronize with the current tempo and introduce slight changes to it gradually, within the allowable margin for tempo adaptation. This produces a beautiful effect of *carrying* the tempo over time. The faster the dancer moves in synchronization with the tempo, the faster the tempo will be, and vice-versa.

On the other hand, the operator of the system can change the margin of adaptation at runtime, so she can decide if the tempo will change or not according to the dancer’s rate, and also determine how radical the change will be. This can be done through passing different values of the allowable margin for tempo variation to **m.clock** via the message ‘delta.’ Values for delta very close to zero will make it harder to change the tempo (the value zero will not allow changes at all). Values

helpful feature in case one wants to start a musical sequence at a certain tempo to be subsequently changed by the dancer.

closer to 1 will allow a large range of values to be considered as a margin for tempo adaptation, and the clock will stop behaving like a clock since virtually any value can be considered a good estimate. As an example, let us suppose that delta has the value of 1 and the current tempo estimate is 750 ms. The margin for tempo adaptation will range from $750 - (1 \cdot 750)$ to $750 + (1 \cdot 750)$, i.e. 0 to 1500 ms, which exceeds the range of musical beats.

m.clock still offers other possibilities for a composer to give a dancer several degrees of control of musical tempo in real time. Besides utilizing the message 'delta' to change the allowable margin for adaptation, the user can also reinitialize the object at runtime with different values for the initial tempo estimate through the message 'accept.' This can also be used to optimize the **m.clock**'s initial estimate of the tempo. Since the clock, when not initialized, will accept the first value from **m.bandit** that is between musical beat boundaries, if the value is a 'noisy' measurement (for example, caused by a movement inflexion that outputs a completely different value from what is expected to be the tempo) it can be very hard for **m.clock** to retrieve the real tempo value. The message 'accept' can also be used to initialize the clock with a value at runtime.

6. EXAMPLES OF APPLICATION

I shall now present two examples of application of these objects in interactive dance performance — generation of musical rhythmic structures, and control of musical tempo from dance movement. The examples are presented in a generic fashion, as the application of these techniques in a real performance situation requires that different strategies may be used according to the specific circumstances of that performance.

6.1. Generating Rhythmic Structures from Dance Movement

Besides outputting the fundamental frequency of the frame-differencing signal, **m.bandit** outputs a list in the form of $f_1 a_1 f_2 a_2 f_3 a_3 f_4 a_4$ from its second outlet, in which f and a correspond respectively to the frequency and amplitude of the harmonics (f_1 is the fundamental). This output from **m.bandit** can generate four independent consonant rhythmic layers, each one corresponding to one harmonic of the signal, which can be used to provide the rhythms for a melody or musical texture. Once this list is unpacked by Max's **unpack** object, one can convert the frequencies to milliseconds to feed the inlet of the **metro** object, and use the amplitude values as thresholds that turn this object on and off. The example below illustrates this situation (Figure 9).

Figure 9 shows a patch fragment in which the harmonic list output of **m.bandit** is used to control four **metro** objects that trigger MIDI note on events. To produce output given above, I used a theoretical signal of fundamental 1 Hz containing three harmonics at different intensities (however, in a performance situation, the object's inlet should be receiving the amount of motion as computed by a frame-differencing

algorithm). The rate at which each **metro** object outputs the 'bang' messages is the frequency of each harmonic converted to milliseconds by the object **hz2ms** — a subpatch that converts Hertz to milliseconds. Below the amplitude of each harmonic, the object **>** (greater than) has the threshold value as argument. This is used to turn on or off the **metro** object. The note on message is only triggered when its corresponding harmonic amplitude has passed the threshold value.

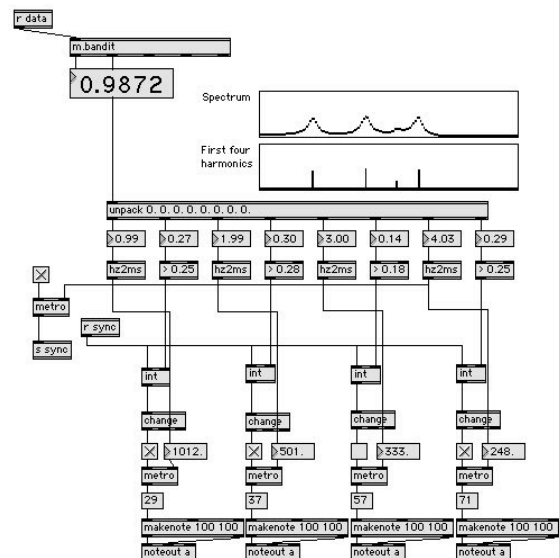


Figure 9. Musical rhythm generation using **m.bandit**.

Generating rhythms directly from the output of **m.bandit** — by utilizing the amplitude and frequency of the harmonics of the signal to control the rate and triggering of rhythms — can produce very interesting effects in the real-time interaction between dance movement and music. Since the fundamental frequency varies at a rate that is proportional to the movement, a waving effect in the rhythm that accompanies the dancer's movement can be heard. This produces a very organic relationship between rhythm and movement.

6.2 Controlling Musical Tempo from Dance Movement

For the reasons explained above, if we want to give the control over musical tempo to a dancer, the fundamental frequency output by **m.bandit** should be converted to milliseconds and sent to **m.clock**. The patch in Figure 10 shows a generic patch fragment illustrating how **m.clock** can be initialized with a tempo estimate (message 'accept 700') and how the message 'delta' can be sent to this object in order to gradually allow the dancer to change the tempo of the musical sequence being played.

If a sequence is playing with a tempo of 700 ms and we want to gradually give the control of that tempo to a dancer, one should gradually increase the value for delta in order to enable the tempo change.

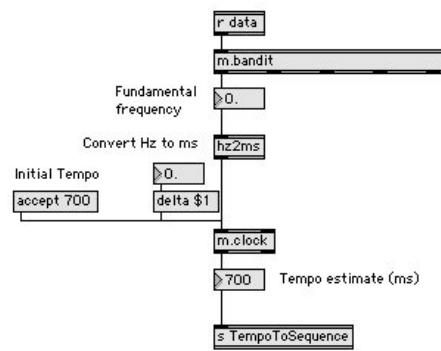


Figure 10. Patch fragment illustrating the use of **m.clock** and its messages.

7. CREATIVE WORK DONE SO FAR

I utilized this m-objects library in three creative efforts so far: *Etude for Unstable Time* (2003-2005), *Olivia* (2004), and *Will.0.w1sp* (2005).

Etude for Unstable Time was the first piece for interactive dance created using the m-objects in 2003 in collaboration with choreographer/dancer Maxime Iannarelli and was premiered publicly in Pisa at the PLAY! concert of the *Computer Music Modeling and Retrieval 2005* conference on September 26. This short piece consists of a structured improvisation in two parts. In the first part, the rhythm of music is generated solely by the computer's rhythmic interpretation of dancer's movement. In the second part there is a game played between the dancer and me over the control of the tempo of sequence of samples being played.

The m-objects had their first public presentation in *Olivia*, a dance solo for children based on the cartoon character created by Ian Falconer. This solo was choreographed and danced by Isabel Barros, with music composed by me, and was premiered at Teatro Rivoli, in Porto, Portugal, on May 6, 2004. In this piece, I used the m-objects to create two *magical moments* in the show. The first was when Olivia went to the museum and pictured herself in a ballet class. When the dancer started dancing, a piano solo was generated utilizing the rhythms produced by her movement, thereby giving the impression that the character Olivia was controlling the piano music. The second *magical moment* happened when Olivia was having insomnia at night and started rolling in bed until eventually began to dance, generating all of the rhythm being played by her movement.

Will.0.w1sp is an interactive installation conceived by Kirk Woolford in which a visual particle system is generated from captured sequences of dance movement, premiered at The Waag Society in Amsterdam on March 16, 2005. In this installation, the m-objects interpret the movement of the visual particles in order to generate the rhythms that control the sonic grains of the soundscape.

8. MORE INFORMATION

The m-objects and other related information are available for download at <http://homepage.mac.com/carlosguedes>.

9. ACKNOWLEDGEMENTS

This software was developed as part of my doctoral dissertation [8]. I would like to thank George Fisher, Robert Rowe, and Ann Axtmann, my committee members for all their support and guidance. Part of this research was also developed at the Institute of Sonology in the Hague, and I would like to thank Peter Pabon for his guidance in the development of **m.bandit**. I would like to thank Ali Taylan Cemgil for his ideas and criticism on the implementation of **m.clock**. My PhD studies at NYU were kindly supported by the Foundation for Science and Technology (grant BD 13934/97) and the Luso-American Foundation in Portugal. The research done at the Institute of Sonology was partially funded by NUFFIC, The Netherlands Organization for International Cooperation in Higher Education.

10. REFERENCES

- [1] Banks, K. *The Goertzel Algorithm*, 2002, <http://www.embedded.com/showArticle.jhtml?articleID=9900722>, retrieved March 8, 2005
- [2] Blahut, R. *Fast algorithms for digital signal processing*. Addison-Wesley Publishing Company, Reading, MA, 1985
- [3] Cemgil, A. T., Kappen, H. J., Desain, P., & Honing, H. "On tempo tracking: Tempogram representation and kalman filtering," *Proceedings of the International Computer Music Conference* (pp. 352-355), Berlin, Germany, 2000
- [4] Fraisse, Paul. "Rhythm and tempo." In D. Deutsch (Ed.), *The psychology of music* (pp. 149-180). Academic Press, London, 1982
- [5] Fraisse, Paul. *Les structures rythmiques*. Publications Universitaires de Louvain, Louvain, France, 1956
- [6] Goertzel, G. "An algorithm for the Evaluation of Finite Trigonometric Series," *American Mathematics Monthly*, 65 (pp. 34-35), 1958
- [7] Guedes, C. "Controlling musical tempo from dance movement: A possible approach," *Proceedings of the International Computer Music Conference* (pp. 453-457), Singapore, 2003
- [8] Guedes, C. *Mapping Movement to Musical Rhythm: A Study in Interactive Dance*. Ph.D. Thesis, New York University, New York, NY, 2005
- [9] Guedes, C. "The m-objects: A Small Library for Musical Rhythm Generation and Musical Tempo Control from Dance Movement in Real Time," *Proceedings of the International Computer Music Conference* (pp. 794-797), Barcelona, Spain, 2005
- [10] <http://homepage.mac.com/davidrokeby/softVN/S.html>
- [11] <http://www.cycling74.com>

- [12] <http://www.cycling74.com/downloads/cyclops>
- [13] Large, E., & Kolen, J. "Resonance and the perception of musical meter," *Connection Science*, 6(1) (pp.177-208), 1994
- [14] Openheim, A., & Schafer, R. *Digital signal processing*. Prentice-Hall, Englewood Cliffs, NJ, 1975
- [15] Pabon, P. Personal communication, October 31, 2002
- [16] Parncutt, R. "The perception of pulse in musical rhythm." In A. Gabrielsson (Ed.), *Action and Perception in Rhythm and Music* (pp. 127-138). Royal Swedish Academy of Music, Stockholm, 1987
- [17] Parncutt, R. "A perceptual model of pulse salience and metrical accent in musical rhythm," *Music Perception*, 11 (pp. 409-464), 1994
- [18] Rowe, R. *Machine Musicianship*. MIT Press, Cambridge, MA, 2001
- [19] Scheirer, E. "Tempo and beat analysis of acoustic musical signals," *Journal of the Acoustical Society of America*, 103(1) (pp. 588-601), 1998
- [20] Sirin, S. *A DSP algorithm for frequency analysis*, 2004, <http://www.embedded.com/showArticle.jhtml?articleID=17301593>. Retrieved March 8, 2005
- [21] Tekalp, M. *Digital video processing*. Prentice-Hall, Englewood Cliffs, NJ, 1995
- [22] Tempelaars, S. *Signal processing, speech and music*. Swets & Zeitlinger, Lisse, The Netherlands, 1996
- [23] Foiviainen, P. "An interactive MIDI accompanist," *Computer Music Journal*, 22(4) (pp. 63-75), 1998