Decentralized Harmonic Synchronization in Mobile Music Systems

Kristian Nymoen
Department of Musicology,
University of Oslo, Norway
Email: kristian.nymoen@imv.uio.no

Arjun Chandra, Kyrre Glette and Jim Torresen
Department of Informatics,
University of Oslo, Norway
Email: {chandra, kyrrehg, jimtoer}@ifi.uio.no

***Abstract*—A system for decentralized synchronization of musical agents is presented, inspired by Mirollo and Strogatz' pulse-coupled oscillator model of the synchronous flashing of certain species of firefly. While most previous work on pulse-coupled oscillators assume fixed and (close to) equal oscillator frequencies, the presented system tackles the challenge of different starting frequencies. Open source implementations in Puredata, Max, and Matlab are provided. Test results for setups of six nodes show that nodes reach a state of *harmonic synchrony*, where fire events coincide and oscillators display integer-ratio frequency relations.**

I. Introduction

Research on interactive music systems has become increasingly popular with the emergence of various mobile technologies. These technologies enable people to consume or perform music anywhere. While portable technologies for music consumption have been widespread since the 1980s, the developments in last decade have also allowed an increased research effort towards developing expressive musical instruments on mobile platforms. Music technologies are often seen as either technologies for creating music or technologies for playing back prerecorded music.

Active music technologies challenge the traditional distinction between musical instruments (used by performers) and music playback devices (used by listeners). The two may be seen as opposite extremes on a continuum (Fig. 1), where technologies allow different degrees of interaction with the music. Active technologies provide users with a higher degree of control than traditional music playback devices, yet not requiring the expertise of professional performers on musical instruments. Examples of such technologies are music games like Guitar Hero [1] and devices that allow controlling musical parameters based on various sensor inputs, e.g. by jogging [2].

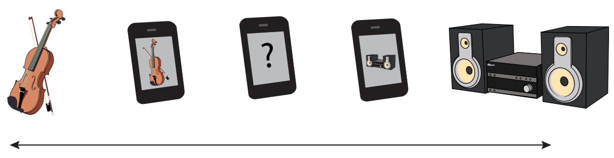

Fig. 1. Active music technologies explore musical interaction between instrument and playback device.

This research has received funding from EU FP7 under grant agreement no. 257906 (EPiCS).

A. Collaborative active music

Our focus of research is on collaborative active music, meaning a group of people who are using their mobile phones to interact with music at a level where the degree of control is higher than traditional media players, but still more constrained than traditional musical instruments. The target group is non-musicians, who may require varying degrees of assistance from their respective mobile device. By applying techniques from artificial intelligence, the devices used in the system may process the control data in such a way that a degree of musical coherence is preserved.

In our work, the system at hand is described as a network of nodes, where each node is a mobile device that is controlled by a human user or by a computational agent. To ensure stability and scalability, we require the system to be *decentralized*, which means that there exists no central point of control in the network. Thus, desired global behavior has to emerge from the actions of and interactions between nodes via algorithms implemented locally on each node. As such, we specify *self-awareness* as a requirement for the nodes [3]. In other words, the nodes need mechanisms for analysing the musical scenario within which they are playing, and mechanisms for adapting their musical output accordingly.

B. Synchronization

Of the many challenging research topics that exist in the implementation of a decentralized active music system, our main focus in the present paper is decentralized *synchronization*. If, at some point in the musical performance, a node detects that it is out of sync with the rest of the group, it must be able to take control of the timing.

In order to tackle the problems of decentralized synchronization of musical agents, we take inspiration from previous research in computational biology and adaptive systems, and in the modeling of perception of musical meter. We present an effective implementation inspired by Mirollo and Strogatz' [4] algorithm for synchronizing the phase of pulse-coupled oscillators (PCO) implemented on mobile devices. To remove the need for any external communication protocol, all communication is done through audio. Each node is able to output short impulsive tones through its loudspeaker and obtain audio data through its microphone. The node is required to extract tone onsets from the audio input, but is unable to distinguish between the output from different nodes.

As will be presented, previous research on PCOs has mainly been concerned with synchronization of nodes with equal, or almost equal, frequencies. We introduce a novel algorithm where both oscillator *phase* and *frequency* are adjusted. This allows initial tempo differences between nodes, converging to a state of what we shall call *harmonic synchrony*, to be discussed further in section III-B1.

Before returning to our own algorithm, we will in the next section present previous work on synchronization of agents in music and in general. In section III we explain the basic concepts of PCO, and our extension to the algorithm. In section IV, implementations in Max, Puredata, and Matlab are presented, followed by experiments and results in section V. Finally, conslusions and plans for future extensions of the work are presented in section VI.

II. Background

We start this section by briefly outlining various previous research on synchronization in music. Then we discuss firefly inspired synchronization mechanisms in general.

A. Autonomous Synchronization in Music

In music research, synchronization has been studied from several perspectives. In music cognition and perception, much effort has been put into studying how people synchronize to a musical beat, e.g. [5], [6], or to each other while engaging in musical activities such as singing or dancing [7], [8]. One model of human synchronization to music we find particularly interesting has been presented by Large and Kolen [9]. Their synchronization mechanism uses bi-directional adjustment of phase and frequency, which has also been of importance in the present paper. We discuss this further in Section III-B4.

A slightly different perspective on synchronization in music research is to program computers in such a way that they are able to follow musical beats. Lambert's *Crickets* system is an example of a system where all nodes adapt in order to agree on a common output pattern [10]. Other systems, such as *beat trackers* [11], adapt to the tempo of a master unit.

Most of the models used in music are systems where nodes synchronize to a master unit. Lambert's Crickets system is an exception, being decentralized, but phase-coupled rather than pulse-coupled. This means that the interaction between oscillators is continuous, making the model very difficult to implement in systems with separate computational nodes [12].

B. Firefly-based synchronization

Researchers have worked on modelling the emergence of synchronization in nature via oscillators since the 1960s [13]. While most early work focused on phase-coupling between oscillators, Mirollo and Strogatz [4], inspired by the work of Peskin, argued that many oscillators in nature are coupled by pulse-like interactions, giving the example of certain species of firefly which adapt their flashing rhythms when observing flashes from other fireflies. Building on Peskin's model, Mirollo and Strogatz presented a PCO model that converges towards synchrony for an arbitrary number of oscillators.

The need for synchronization in decentralized computing systems has triggered the application of the PCO approach in such systems in recent years. Babaoglu et al. used the fireflies approach to synchronize clock cycles of nodes in peer-to-peer networks [14]. Klinglmayr et al. offer an adaptation of the firefly approach to synchronization in dynamic networks. Specifically, targeting the problem of robustness against faulty nodes [15]. The *Reachback Firefly Algorithm* has been proposed in order to deal with the problem of nodes being unable to receive messages or 'flashes' while transmitting [16], [17]. Another proposed solution to this problem is *slot synchronization* [18].

Although most of the work presented above, and indeed most previous work on PCO is concerned with oscillators with equal frequencies, some research efforts have been made toward frequency adaptation of the oscillators. Ermentrout [19] proposed a phase-coupled model where oscillators would adjust their own frequencies within certain fixed boundaries and converge towards both common frequency and phase. However, in addition to the limitations of a phase-coupled system, the boundaries must be defined up front, and the range of starting frequencies from which the system reaches synchrony is limited. Konishi and Kokame [20] presented a model where non-identical pulse-coupled oscillators could also synchronise, given specific ranges for their refractory periods and frequency distributions.

III. Problem Description

An oscillator in our system is part of a self-contained musical unit, hereby called a *node*, which contains several elements: mechanisms for detecting sound onsets from other nodes, self-adaptation, self-assessment, and playing tones. The elements are to be described more in detail in this section. A node may be implemented in an embedded system, such as a mobile phone, or modeled in software. We specify the following requirements for the system:

- The system should be completely decentralized, meaning that there is no central control over the interaction between nodes. This ensures maximum flexibility, allowing nodes to leave or enter the network at any time.
- To remove the need for any external communication protocol, all communication should be done through audio. Each node is able to output short impulsive tones through its loudspeaker, and to obtain audio data through its microphone. The node is required to extract tone onsets from the audio input, but is unable to distinguish between the output from different nodes.
- A node is aware of its current phase, but unaware of the phase of other nodes.
- Temporal components in music tend to appear in an integer-ratio relation to each other (e.g., beats, measures, phrases, or quarter notes, 8ths, 16ths). As such, we do not require all nodes to fire at the same time, but to fire at a common underlying *pulse*. We call this target state *harmonic synchrony*, to be specified in section III-B1.

A. Phase Adjustment in Pulse-Coupled Oscillators

Various descriptions of pulse-coupled oscillators have been made, using slightly different terminology. Our description below uses much of the terminology from [21], however, some adjustments have been made to address the problem at hand.

An oscillator $\imath$ is represented by its *phase*, $\phi_\imath(t)$. The phase is initialized randomly (between 0 and 1), and evolves over time (t) toward 1 at a rate of $\omega_\imath(t) = \frac{d\phi_\imath}{dt}$, this rate is the *frequency* of the oscillator. When the phase of oscillator $\imath$ reaches maximum, the node "fires" by playing a short tone, and resets back to 0 before continuing to evolve toward 1.

In the case when the frequencies of the oscillators are identical (or nearly identical) a node performs self adaptation by updating its phase only. Each time a node $\imath$ perceives a fire event from a node $\jmath$, it immediately increases its own phase by some amount. This amount is defined by the phase update function, $P(\phi_\imath(t))$. More precisely:

$$\phi_\jmath(t) = 1 \Rightarrow \begin{cases} \phi_\jmath(t^+) = 0 \\ \phi_\imath(t^+) = P(\phi_\imath(t)) \quad \forall \imath \neq \jmath \end{cases} , \tag{1}$$

where t^+ denotes the time step immediately after t. The phase update function is given by:

$$P(\phi) = (1+\alpha)\phi, \tag{2}$$

where α is the *pulse coupling constant*, denoting the coupling strength between nodes.

Mirollo and Strogatz' evidence for synchronization of pulse-coupled oscillators assumes that communication between nodes is done by infinitely short impulses without transmission delay. Real systems contain delays, and the tones played by the nodes in our system are not infinitely short. To cope with this, a *refractory period* (t_{ref}) is introduced immediately after each firefly has fired [22]. During this period the oscillator is prevented from adjusting its state. Figure 2 shows the interaction between two nodes with constant and equal frequencies.

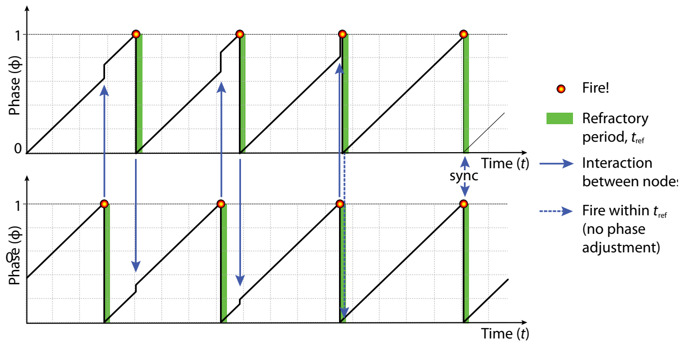

Fig. 2. The figure illustrates how two equal pulse-coupled oscillators synchronize using Mirollo-Strogatz algorithm.

B. *Frequency Adjustment in Pulse-Coupled Oscillators*

When nodes are not assumed to have fixed and equal frequencies, synchronization is more difficult. In order to solve this problem we propose a new synchronization mechanism. The mechanism incorporates a model of self-awareness, in which nodes assess their own level of synchrony. A small change in the phase update function was also necessary to solve the frequency synchronization problem. The mechanism is presented in more detail below, but first we shall present the target state of our system: *harmonic synchrony*.

1) Harmonic Synchrony: As opposed to traditional pulse-coupled oscillators, all nodes in our system are not required to fire simultaneously. Being an interactive music system, people may want their device to synchronize with different subdivisions of a measure (e.g. some play quarter notes while others play 8ths). We choose the term *harmonic synchrony*, taken from the concept of *harmonics* in the frequency spectrum of a waveform, where the frequency of every harmonic is an integer multiple of the lowest (fundamental) frequency. In other words, harmonic synchrony is a state where the frequency of each node is element of $\omega_{low} \cdot 2^{\mathbb{N}_0^+}$, where ω_{low} is the lowest frequency of all nodes in the group. Our approach for reaching this state has been to make nodes only fire on every other period, and to implement update functions for phase and frequency which cause (close to) 0 change half-way through the cycle.

2) Self-awareness in synchronizing fireflies: Lewis et al. identified several key properties of what they call *self-aware nodes* in computing systems [3]. Primarily, to be self-aware, they conclude that a node must

- Possess information about its internal state.
- Possess sufficient knowledge of its environment to determine how it is perceived by other parts of the system.

In the context of synchronizing firefly imitating nodes, these key properties are reflected by a node's awareness of its current phase, and its ability to assess its level of synchrony with the received fire events from other nodes. Specifically, each time a node detects a fire event from another node, an error-measure, $\epsilon \in [0,1]$ is calculated, which is at its highest value when $\phi = 0.5$, and lowest value when ϕ is equal to 0 or 1. More precisely, we let

$$\epsilon = sin(\pi\phi(t))^2 \tag{3}$$

with the special case that $\epsilon = 0$ if a fire event is received within the refractory period. We may thus let $\epsilon(n)$ be a discrete function describing the error measures at the n-th fire event received by a node. We calculate the self-assessed synch score of each node, s, by applying a running median filter to $\epsilon(n)$:

$$s = \text{median}\{\epsilon(n), \epsilon(n-1), ..., \epsilon(n-m)\}, \tag{4}$$

where $m-1$ is the length of the median filter. Thus, s takes a high value when the node is out of phase with the past received fire events, and a low value when the node is in phase with the past received fire events. The use of the self-assessment measurement in the frequency update function is covered in the section below.

3) Frequency update function: The only information a node has to rely on for frequency adjustment are the discrete sound onsets from other nodes. We specify a function that decreases frequency if a fire event is received in the first half of its cycle, and speeds up if in the last half (to "catch up" with the firing node). Upon receiving a fire event from another node, the receiving node calcultates ρ, which is negative when $\phi < 0.5$ and positive when $\phi > 0.5$.

$$\rho = \sin(2\pi\phi(t)) \tag{5}$$

We let $\rho(n)$ be a discrete function denoting the amplitude and sign of frequency modification of the n-th received fire event (cf. Equation 5), and $s(n)$ describe the current self-assessed synch score for the same fire event (cf. Equation 4). We let $H(n)$ describe the product of the two, indicating of how a node should adjust its frequency based on the n-th fire event received.

$$H(n) = \rho(n)s(n), \tag{6}$$

Note that H outputs a value between -1 and 1 indicating whether ω should be decreased or increased. In our first experiments, where this function was applied to immediately change the frequency of a node, the system often converged to $\omega = 0$. To prevent this, we take use of Werner-Allen's *reachback firefly algorithm* (RFA) [16]. Originally designed for phase updates with the purpose of preventing "deafness" in a firefly system, the concept of RFA is useful also in frequency updates. RFA specifies a system which, rather than making immediate phase jumps upon received fire events, collects the received fire events and applies the total phase jump at the beginning of its next cycle. We apply the same principle to frequency updates, as illustrated in Figure 3 for the case when a single fire event is received within a period. More precisely, frequency updates are performed in the following manner:

$$\phi_\imath(t) = 1 \Rightarrow \begin{cases} F(n) & = \beta \cdot \sum_{x=0}^{y-1} \frac{H(n-x)}{y} \\ \omega_\imath(t^+) & = \omega_\imath(t) \cdot 2^{F(n)} \end{cases}, \tag{7}$$

where β is the *frequency coupling constant*, and y is the number of received fire events during the latest oscillator period. The maximum value of $\omega_\imath(t^+)$ is $2\omega_\imath(t)$, and happens only when $\phi = 0.25$ and $\beta = s = 1$. Similarly, the minimum value of $\omega_\imath(t^+)$ is $\frac{1}{2}\omega_\imath(t)$, and happens only when $\phi = 0.75$ and $\beta = s = 1$.

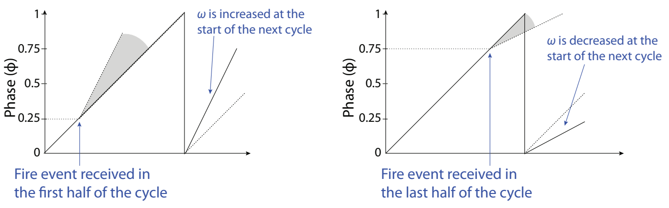
1
0.75
0.5
0.25
0
Phase (ϕ)
ω is increased at the start of the next cycle
Fire event received in the first half of the cycle
ω is decreased at the start of the next cycle
Fire event received in the last half of the cycle

Fig. 3. The figure displays how frequency of a node is adjusted when a single fire event is received during the period. The grey areas show the range of possible adjustment of ω for $\phi = 0.25$ and $\phi = 0.75$, respectively. Phase adjustment is left out of this example.

4) Bi-directional phase shifts: Klinglmayr et al. [15] have showed that the use of both excitatory and inhibitory phase couplings between oscillators may be beneficial. That is, perform a negative phase jump when a fire event is received during the first half of the cycle, and a positive phase jump when a fire event is received in the last half of the cycle. A similar approach was taken previously by [9], who used a sinusoidal delta-function to adjust the oscillator phase. Our experiments showed that this was beneficial in our system, and so we change our phase coupling function accordingly. Due to the target of harmonic synchrony, we require a small period half-way through the oscillator cycle in which no (or only negligibly small) phase jumps are made. This is achieved by our changed phase update function, P:

$$P(\phi) = \phi - \alpha \cdot \sin 2\pi\phi \cdot |\sin 2\pi\phi| \tag{8}$$

The function causes an oscillator to make a phase shift towards 0 or 1 (whichever is the closest) when a fire event is received. Phase jumps half-way through the cycle are attenuated by the $|\sin 2\pi\phi|$ factor.

5) Other frequency adjustment mechanisms: In addition to the main frequency adjustment mechanism presented above, a few other mechanisms are included in our system, mainly as mechanisms to prevent the system from reaching unstable states. We set maximum and minimum frequency thresholds for each node. If ω moves outside these thresholds, a new ω value is set randomly. The lower threshold in our system is set to 0.5 Hz, and upper threshold to 8 Hz. Our results presented in section V show that the node frequencies rarely move outside these limits.

When oscillators with large frequency differences interact, the low-frequency node run the risk of continuously being reset to 0, not ever reaching beyond $\phi = 0.5$. To prevent this, a node is programmed to double its own frequency if its phase is reset repeatedly without ever reaching maximum. Additionaly, as mentioned previously, our approach differs from previous approaches by having nodes fire only every other time when maximum phase is reached.

IV. Implementation

The system has been prototyped in Max[1] where a node is represented by a Max patch which is able to send and receive audio signals to and from a common channel. By opening several instances of this patch, we simulate several fireflies within audible range of each other. Parts of the system has also been implemented in Puredata (PD),[2] which makes it possible to run the algorithm on mobile devices (Figure 4). A Matlab simulation has been set up to enable faster testing of different parameter settings for varying numbers of nodes. The developed software is open source and available online.[3]

Fig. 4. Picture of three iOS devices running the firefly Puredata patch

A. Prototyping in Max

The Max patch contains five main elements. (1) A *listener*, detecting onsets in the input audio stream, (2) an *oscillator*, which outputs a ramp signal between 0 and 1 at some randomly initialized rate, (3) a *phase-adjustment patch*, adjusting the phase of the oscillator, (4) a *frequency-adjustment patch*, adjusting the frequency of the oscillator, (5) a *synthesizer*, generating short sounds when the oscillator reaches maximum. A flowchart of the system is displayed in Figure 5.

[1]http://www.cycling74.com
[2]http://puredata.info
[3]http://fourms.uio.no/downloads/software/musicalfireflies/

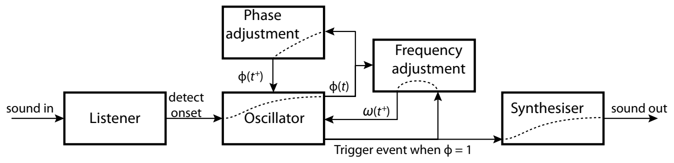

Fig. 5. Schematic overview of the structure of the firefly Max patch.

When a fire event is detected by the listener, it initiates calculation of phase adjustment and frequency adjustment of the oscillator. Phase adjustments are performed immediately, and frequency adjustments are 'collected' and performed when the oscillator reaches maximum. The synthesizer is based on FM synthesis, set to output a random note from a pentatonic scale (C4, D4, E4, G4, A4). The tones are generated with an impulsive dynamic envelope to allow easy onset detection (rise-time 6 ms, decay-time 300 ms). In addition to the functional elements, a visualisation of each node was created, showing a drawing of a firefly whose tail lights up upon firing.

B. *PD/iOS implementation*

Parts of the synchronization system has been implemented in Puredata which facilitates porting the algorithm to mobile devices. We use the iOS application *MobMuPlat* [23] to run the PD patch on iOS devices. The MobMuPlat application is only able to run components from the most basic implementation of PD (known as PD vanilla), which complicates the process of porting the system from Max. At the time of writing, the oscillator, listener, synthesizer and phase-adaptation parts are implemented. We are working on also incorporating the frequency adaptation in the PD implementation. A video of the PD patch running on six iOS devices is available online.[4]

C. *Matlab simulation*

The system has also been implemented as a Matlab function. The user may change a range of variables, such as α, β, number of nodes, and more. The matlab function allows plotting the phase and frequency of all of the nodes, and an animation of how the phase and frequency of all the nodes evolve over time. A video showing this animation is available online.[5] The delays in a real system is difficult to model in the non-realtime Matlab implementation. As such, the Matlab function is useful for initial testing, plotting, and animation, but tests with the real-time implementations in Max and PD provide more reliable results for a real system.

V. Experiments and Results

Test environments have been set up in Matlab and Max. Evaluation has been done by measuring synchronization times and success rates for various parameter settings.

A. *Synchronization measure*

In order to evaluate the synchronization times and success rates of various setups, a measure must be defined for when the system has reached a state of synchrony. Using the firings of the nodes, a set of conditions that had to be met were defined:

- Firing may only happen within a short time period t_f.
- Between each t_f, a period t_q without fire events must be equally long k times in a row.
- All nodes must have fired at least once during the evaluation period.

These rules are best illustrated visually, as in Figure 6. Our own qualitative assessment of the nodes being synchronised coincided with the parameters $t_f \leq 80$ms and $k \geq 8$, and as such these values were used in the experiment. t_q depends on the frequency to which the nodes converge, and is therefore not specified.

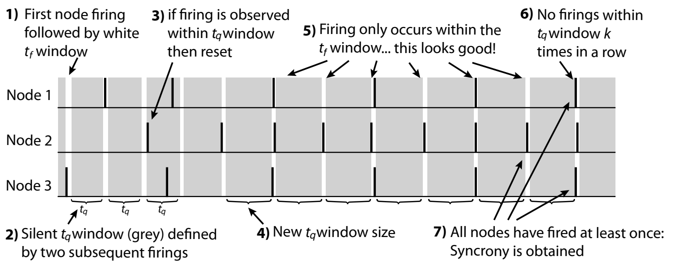
1) First node firing followed by white t_f window
3) if firing is observed within t_q window then reset
5) Firing only occurs within the t_f window... this looks good!
6) No firings within t_q window k times in a row
Node 1
Node 2
Node 3
t_q
t_q
t_q
2) Silent t_q window (grey) defined by two subsequent firings
4) New t_q window size
7) All nodes have fired at least once: Syncrony is obtained

Fig. 6. Illustration of the synchronization measure used in testing of the system. Three nodes are shown, with a black vertical line indicating a fire event. The grey areas denote the t_q period, and the shorter t_f areas denote the windows within which the nodes are allowed to fire.

B. *Phase synchronization*

We started by evaluating the performance of the system in a simple phase synchronization task in Max. The node phases were randomly initialized. A range of values of α was tested, with 30 runs per value. The frequencies of all nodes were set equal ($\omega = 1$ Hz) and fixed (i.e. $\beta = 0$), and the refractory period was set to 50 ms. Communication between nodes occurred only through audio, but additionally, each node sent a message to an external evaluation patch where the fire events of all the nodes were recorded. This patch also acted as an observer, terminating the current run if one of the two following conditions were met:

- a state of synchrony was reached
- synchrony was not reached within 5 minutes

The results from the phase synchronization task are shown in Figure 7. Overall, the experiment showed little difference between synchronization times for different values of α, with a small tendency towards longer synchronization times for lower α. All runs reached synchrony well within the time limit of 5 minutes. The mean synchronization time for all runs and all α-settings was 10.9 s.

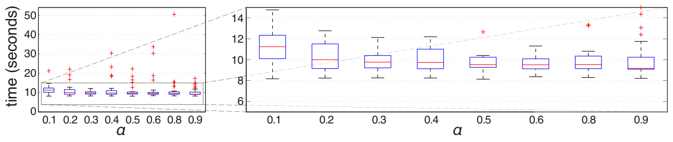

Fig. 7. Synchronization times for six nodes with equal and fixed frequency (1 Hz). 30 tests were performed per α. Only small differences were observed. The full range is shown on the left with a more detailed view on the right.

[4]http://vimeo.com/67205605
[5]http://vimeo.com/72493268

C. Frequency synchronization

Synchronization of nodes with different starting frequencies is more challenging — to the point that not every run reaches synchrony within the time limit. A range of values for the phase and frequency coupling strength was tested on six nodes.

Figure 8 shows the frequency synchronization results for our experiment ordered by α value. The same, ordered by β, is show in Figure 9. The figures show that most runs reach a synchronous state within the time limit, and suggest that low α levels may prevent the system from reaching synchrony. Among the successful runs, most reached synchrony in less than thirty seconds (mean: 22.4 ± 31.3 seconds). The time, however, should not be given too much attention, since the time greatly depends on the frequency to which nodes converge. When the frequency is higher, nodes are able to make more adjustments within a shorter timespan.

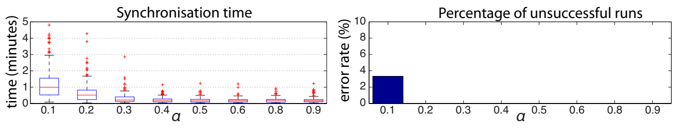

Fig. 8. Frequency synchronization scores ordered by phase coupling strengths (α). When $\alpha = 0.1$ sync times are long, and the experiment also shows an error rate of 3.3 %. Increasing α improves the sync times. No improvement was seen beyond $\alpha \approx 0.4$.

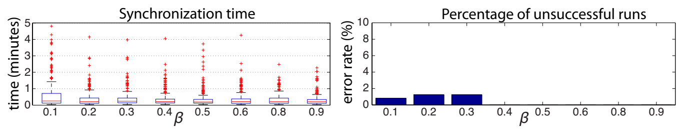

Fig. 9. Frequency synchronization scores ordered by different frequency couplings (β). Except for a few unsuccessful runs for low β values, the tested values for the beta constant gave similar results.

VI. Conclusions and Future Work

A system for decentralized synchronization of musical nodes has been presented. The nodes communicate solely through audio, which simplifies adding or removing nodes from the network. Self-assessment of synchrony, bidirectional phase shifts and a variant of the reachback firefly algorithm together contribute to the system converging to a state of harmonic synchrony. Implementations for Max, Puredata, and Matlab have been made available. To the best of our knowledge, the research introduced in this paper is the first example of a fully decentralized interactive music system in which nodes are communicating solely through audio, where the use of pulse-coupled oscillators allow stigmergic synchronization of both phase and frequency.

Network topologies have not been considered in this paper, and the presented experiments assume that all nodes are able to receive fire events from all of the other nodes. In the future, we will investigate the system performance with different spatial distribution between nodes.

References

[1] "Guitar hero" (software), Cambridge, MA: Harmonix / Red Octane, 2005.

[2] B. Moens, L. van Noorden, and M. Leman, "D-jogger: Syncing music with walking," in *Proceedings of the Sound and Music Computing Conference*, 2010, pp. 451–456.

[3] P. R. Lewis, A. Chandra, S. Parsons, E. Robinson, K. Glette, R. Bahsoon, J. Torresen, and X. Yao, "A survey of self-awareness and its application in computing systems," in *Proceedings of the International Conference on Self-Adaptive and Self-Organizing Systems Workshops (SASOW)*, 2011, pp. 102–107.

[4] R. E. Mirollo and S. H. Strogatz, "Synchronization of pulse-coupled biological oscillators," *SIAM J Appl Math*, vol. 50, no. 6, pp. 1645–1662, 1990.

[5] B. H. Repp, "Sensorimotor synchronization: A review of the tapping literature," *Psychon B Rev*, vol. 12, no. 6, pp. 969–992, 2005.

[6] B. H. Merker, G. S. Madison, and P. Eckerdal, "On the role and origin of isochrony in human rhythmic entrainment," *Cortex*, vol. 45, no. 1, pp. 4–17, 2009.

[7] T. Himberg and M. Thompson, "Group synchronization of coordinated movements in a cross-cultural choir workshop," in *Proceedings of the Conference of European Society for the Cognitive Sciences of Music*, 2009, pp. 175–180.

[8] P. Toiviainen, G. Luck, and M. R. Thompson, "Embodied meter: Hierarchical eigenmodes in music-induced movement," *Music Perception*, vol. 28, no. 1, pp. 59–70, 2010.

[9] E. W. Large and J. F. Kolen, "Resonance and the perception of musical meter," *Connection science*, vol. 6, no. 2-3, pp. 177–208, 1994.

[10] A. Lambert, "A stigmergic model for oscillator synchronisation and its application in music systems," in *Proceedings of the International Computer Music Conference*, vol. 247–252, 2012.

[11] R. B. Dannenberg, "Following an improvisation in real time," in *Proceedings of the International Computer Music Conference*, 1987, pp. 241–248.

[12] I. Bojic, T. Lipic, and V. Podobnik, "Bio-inspired clustering and data diffusion in machine social networks," in *Computational Social Networks*. Springer, 2012, pp. 51–79.

[13] A. T. Winfree, "Biological rhythms and the behavior of populations of coupled oscillators," *J. Theor. Biol.*, vol. 16, no. 1, pp. 15–42, 1967.

[14] O. Babaoglu, T. Binci, M. Jelasity, and A. Montresor, "Firefly-inspired heartbeat synchronization in overlay networks," in *Proceedings of the International Conference on Self-Adaptive and Self-Organizing Systems.*, 2007, pp. 77–86.

[15] J. Klinglmayr, C. Kirst, C. Bettstetter, and M. Timme, "Gurarantееing global synchronization in networks with stochastic interactions," *New J. Phys.*, vol. 14, 2012.

[16] G. Werner-Allen, G. Tewari, A. Patel, M. Welsh, and R. Nagpal, "Firefly-inspired sensor network synchronicity with realistic radio effects," in *Proceedings of the International Conference on Embedded networked sensor systems*, 2005, pp. 142–153.

[17] R. Leidenfrost and W. Elmenreich, "Firefly clock synchronization in an 802.15.4 wireless network," *EURASIP J Embed Syst*, vol. 2009, pp. 1–17, 2009.

[18] A. Tyrrell, G. Auer, and C. Bettstetter, "Emergent slot synchronization in wireless networks," *IEEE Trans Mobile Comput*, vol. 9, no. 5, pp. 719–732, 2010.

[19] B. Ermentrout, "An adaptive model for synchrony in the firefly pteroptyx malaccae," *J Math Biol*, vol. 29, no. 6, pp. 571–585, 1991.

[20] K. Konishi and H. Kokame, "Synchronization of pulse-coupled oscillators with a refractory period and frequency distribution for a wireless sensor network," *Chaos*, vol. 18, no. 3, p. 033132, 2008.

[21] J. Klinglmayr and C. Bettstetter, "Self-organizing synchronization with inhibitory-coupled oscillators," *ACM Trans Auton Adap*, 2012.

[22] R. Mathar and J. Mattfeldt, "Pulse-coupled decentral synchronization," *SIAM J Appl Math*, vol. 56, no. 4, pp. 1094–1106, 1996.

[23] D. Iglesia, "Mobmuplat" (iOS application), Iglesia Intermedia, 2013.