The Challenge of Decentralised Synchronisation in Interactive Music Systems

Kristian Nymoen, Arjun Chandra, and Jim Torresen
Department of Informatics
University of Oslo, Norway
Email: {krisny, chandra, jimtoer}@ifi.uio.no

***Abstract*—Synchronisation is an important part of collaborative music systems, and with such systems implemented on mobile devices, the implementation of algorithms for synchronisation without central control becomes increasingly important. Decentralised synchronisation has been researched in many areas, and some challenges are solved. However, some of the assumptions that are often made in this research are not suitable for mobile musical systems. We present an implementation of a firefly-inspired algorithm for synchronisation of musical agents with fixed and equal tempo, and lay out the road ahead towards synchronisation between agents with large differences in tempo. The effect of introducing human-controlled nodes in the network of otherwise agent-controlled nodes is examined.**

***Keywords*-Pulse coupled oscillators, fireflies, interactive music, synchronization**

I. Introduction

Research on interactive music systems has become increasingly popular with the emergence of so-called ubiquitous computing. Mobile technologies allow people to consume or perform music anywhere. While portable technologies for music playback have been widespread since the 1980s, the developments in last decade have also allowed an increased research effort towards developing novel musical instruments on mobile platforms. Traditionally, a distinction has been made between the *performer*, creating the music, and the *perceiver*, receiving the music. *Musical instruments* are used by performers, and allow a high degree of control of the musical output. Correspondingly, *music players* are used by perceivers, allowing simple controls such as play, pause, skip, and turning the volume up and down.

Active music technologies challenge the traditional distinction between musical instruments and music players. The two may be seen as two extremes on a continuum, where technologies along the continuum allow different degrees of interaction with the music. Active music technologies provide users with a higher degree of control than traditional music players, yet not requiring the expertise of professional performers on musical instruments. Examples of such technologies are music games [1], composition software [2], devices that allow controlling musical parameters based on various sensor inputs, e.g. by jogging [3], and also a large variety of apps for mobile phones allowing people to interact with music anywhere (e.g. [4]).

A. Collaborative active music

Our focus of research is on collaborative active music, here referring to 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 restricted than traditional musical instruments. By allowing users to control the devices, while at the same time retaining some degree of control to be held by an adaptive algorithm in the device itself, users with less musical training are enabled to participate in a collaborative active music experience. We have previously shown that some degree of "musicality" can be preserved in a band made up of of non-musicians, by applying an economics-inspired approach to assist the circulation of solos when a group of non-musicians are playing together [5].

We describe collaborative active music systems 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 maximum flexibility, allowing anyone to enter or leave the network at any time, we require the system to be *decentralised*, which means there exists no central point of control in the network. Thus, desired global behaviour 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 [6], implying a need for mechanisms for analysing the musical scenario within which the nodes are playing, and mechanisms for adapting their musical output accordingly.

B. Synchronisation

Many challenging research topics exist in the scenario we have laid out thus far. This particular paper addresses the problem of decentralised *synchronisation* of musical agents. Synchronisation is a so-called *protomusical* behaviour, meaning a behaviour that exhibits musical features, such as harmonic oscillations or rhythmic patterns, but lacking cultural realisation as music [7]. As such, development of agents able to exhibit protomusical behaviours like musical synchronisation is an important step in the development of decentralised collaborative music systems.

In order to tackle the problems of decentralised synchronisation of musical agents, we take inspiration from previous research in computational biology and adaptive systems. We present an effective implementation of Mirollo and Strogatz' firefly-inspired algorithm for synchronising the phase of pulse-coupled oscillators implemented on mobile devices [8]. 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 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.

II. Background

Theoretical research on modelling the emergence of synchronisation in nature via oscillators has been around since the 1960s [9]. While most early work focused on smooth couplings between oscillators, Mirollo and Strogatz, inspired by the work of C.S. Peskin, argued that many oscillators in nature are coupled by pulse-like interactions, giving the example of certain species of fireflies which adapt their flashing rhythms when observing flashes from other fireflies [8]. Building on Peskin's model to admit more dynamics, Mirollo and Strogatz presented a pulse-coupled oscillator model that converges towards synchrony for an arbitrary number of oscillators.

The need for synchronisation in decentralised computing systems has triggered the application of the pulse-coupled oscillator approach in such systems in recent years. Research efforts have been seen in the field of peer-to-peer networks where peers or nodes need to synchronise their clock cycles in order to efficiently carry out tasks that involve timely communication with other nodes [10], and in wireless networks where the idea is for nodes to have synchronized sleep schedules in order to reduce the power consumption in the network [11]. The field of artificial intelligence, specifically distributed robotic systems, have also found it useful to consider the pulse-coupled oscillator framework to dealing with synchronisation in robotic swarms [12]. Klinglmayr et al. target the problem of robustness against faulty nodes, e.g. nodes that become defective, or malicious intruding nodes, that may disturb the operation of the network [13]. While the pulse-coupled oscillator framework predominantly considers excitatory coupling, in that, the phase adjustments at the receiving nodes push their phases forward in time, inhibitory coupling (pushing phase backward in time) is shown to help against faults.

A. Attributes of Pulse-Coupled Oscillators

Whether theory or applications, various attributes can characterise the type of distributed synchronisation problem one aims at tackling. Indeed, the pulse-coupled oscillator framework has gained much attention at modeling and tackling such problems with varying degrees of success. Some of these attributes are:

- Type of coupling: the coupling between the oscillators can vary from being a tight all-to-all (e.g. pulses sent received by all) one, to couplings characterised by local interactions in systems with a spatial structure with nodes only able to communicate with local neighbourhoods.
- Heterogeneity: oscillators may have the same frequency in which case they are known as identical, or there may be heterogeneity in the frequencies with which they oscillate.
- Communication medium for coupling: the communication of pulses may be in the form of packets on a network, or may be more physically restrictive, e.g. light or indeed sound/audio signals.
- Decentralisation: there may or may not be a single timing source to synchronise with.

The decentralised synchronisation problem within the musical setting that we consider in this paper, as described in Sections I-A and I-B, can be characterised by a system of pulse-coupled oscillators interacting locally via audio signals, without a timing source to synchronise with, and where the oscillators may or may not be identical.

III. Phase Adjustment in Pulse-Coupled Oscillators

An oscillator *i* in our system is represented by its *phase*, $\phi_i(t)$. The phase is initialised randomly (between 0 an 1), and evolves over time (*t*) toward 1 at a rate of $\omega_i(t) = \frac{d\phi_i}{dt}$, this rate is the *frequency* of the oscillator. When the phase of oscillator *i* reaches maximum, the node "fires" by playing a tone, and resets back to 0 before it continues to evolve toward 1.

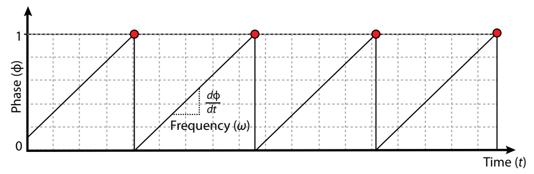


Figure 1. The figure shows the phase ϕ of a single oscillator evolving over time t at rate ω. The red dots indicate that the node fires.

In this section, we assume that all oscillators in the system oscillate at the same frequency, as is also an assumption in Mirollo and Strogatz' work. We define a *phase update function*, $P(\phi_i(t))$, that describes how a node adjusts its own phase upon receiving a fire event from another node. Each time a node *i* perceives a fire event from a node *j*, it immediately increases its own phase by some amount. More precisely:

$$\phi_j(t) = 1 \Rightarrow \begin{cases} \phi_j(t^+) = 0 \\ \phi_i(t^+) = P(\phi_i(t)) \quad \forall i \neq j \end{cases}, \quad (1)$$

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

$$P(\phi) = (1 + \alpha)\phi, \quad (2)$$

where α is a constant denoting the coupling strength between nodes.

Mirollo and Strogatz' evidence for synchronisation of pulse-coupled oscillators assumes that communication between nodes is done by infinitely short impulses without transmission delay. Since our system is communicating through audio, it will inevitably contain delays. To cope with this, a *refractory period*, t_{ref}, is introduced immediately after each firefly has fired [14], [15]. During this period the oscillator is prevented from from adjusting its phase. The process of synchronising the phase of two oscillators is illustrated in Figure 2.

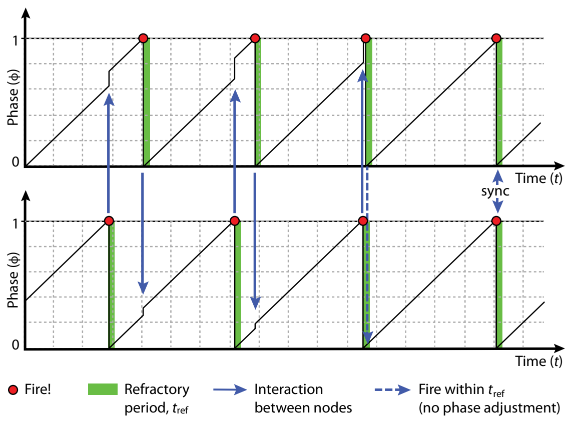


Figure 2. Synchronisation of two pulse-coupled oscillators with equal frequency using Mirollo-Strogatz algorithm and a refractory period.

A. Implementation

Our system has been prototyped in the graphical programming environment Max,[1] where each node is represented by a single Max patch that is able to send and receive audio signals from a common channel. By opening several instances of this patch, we simulate several fireflies within audible range of each other. The Max patch contains four main elements.

1) A *listener*, detecting onsets in the input audio stream.
2) An *oscillator*, oscillating between 0 and 1 with a given frequency and phase.
3) A *phase-adjustment patch*, adjusting the phase of the oscillator.
4) A *synthesiser*, generating short, impulsive sounds when the oscillator reaches maximum.

A flowchart of the system is displayed in Figure 3.

In the prototyping stage, Andrew Robertson's **aubioonset**~ object for Max, based on the *aubio* library by Paul Brossier, has been used for onset detection. Upon perceived activations from other fireflies, the listener initiates calculation of phase adjustment of the oscillator,

[1]http://www.cycling74.com

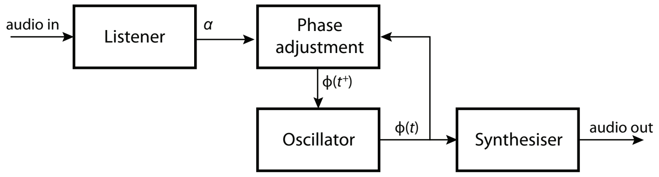
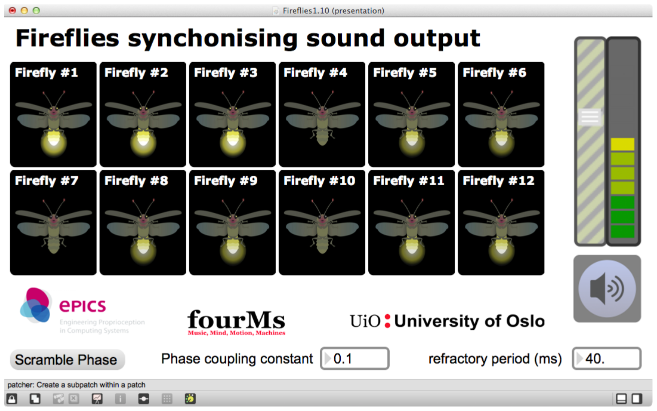


Figure 3. Schematic overview of the structure of the firefly Max patch.

and the oscillator phase is updated accordingly. The synthesiser is based on FM synthesis, set to output a random note from a pentatonic scale (C4, D4, E4, G4, A4). The use of a pentatonic scale ensures a certain degree of harmony between the tones, even when tones are selected randomly. An impulsive dynamic envelope (rise-time 6 ms, decay-time 300 ms) is used to allow easy onset detection. In addition to the functional elements, a visualisation of each node has been created, showing a drawing of a firefly whose tail lights up upon firing.



Phase coupling constant 0.1
refractory period (ms) 40.

Figure 4. Screen shot from a setup in Max with 12 fireflies synchronising.

We have implemented part of the synchronisation system in PureData[2] (PD), which enables the algorithm to be run on other operating systems that those able to install Max. We use the iOS application *MobMuPlat* [16] to run the PD patch on iOS devices. The MobMuPlat application is only able to run components from the most basic distribution of PD (known as PD vanilla), which complicates the process of porting the system from Max. A video of the PD patch running on six iOS devices is available online.[3]

B. Experiments and results

A simple test was set up to evaluate the time needed for the system to synchronise for various α-values and various number of nodes. All nodes were set to fire at 1 Hz, and their phases were randomised at the start of each test run. The refractory period was set to 50 ms. We measured the time from the start until when the overall

[2]http://puredata.info
[3]http://vimeo.com/67205605

system reached a state with all nodes firing within a 50 ms window three times in a row as shown in Figure 5. To minimize variations in synchronisation times caused by differences in the initial state of the system, 30 runs were carried out for each parameter settings for each number of nodes. Figure 6 shows the synchronisation times for three different parameter settings. The overall best results (shown in the middle) were found for $\alpha = 0.1$, where the system would quite often reach a synchronised state within $\sim$10 seconds, even when as many as 30 nodes were involved. A lower value for α drastically increased the times needed to synchronise as shown in the top plot in Figure 6. The bottom plot shows synchronisation times for $\alpha = 0.2$, where the synchronisation times increased with the number of nodes.

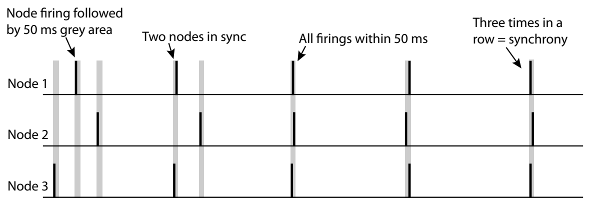

Node firing followed by 50 ms grey area
Two nodes in sync
All firings within 50 ms
Three times in a row = synchrony
Node 1
Node 2
Node 3

Figure 5. The algorithm parameters were evaluated by measuring the time from the initial state to the time when all nodes fired within a 50 ms period three times in a row.

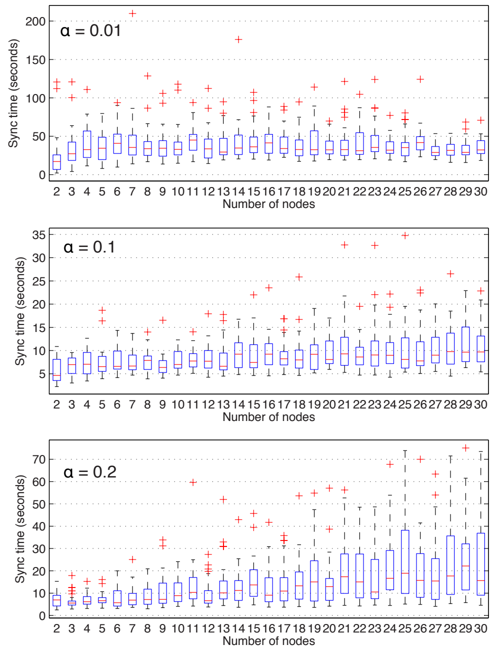


Figure 6. The figure shows the times needed to synchronise for a varying number of nodes using three different parameter settings.

In an interactive music system, autonomous synchronisation is useful for allowing agent-controlled nodes to adjust to the rhythm of human users, or to assist human users who find it difficult to follow the rhythm. A second experiment was set up to evaluate the ability of agent-controlled nodes to adjust to a human user. We allowed a user to override the internally controlled firing of a node by using a shoe with internal force sensing resistors. By tapping his foot out of phase with the rest of the nodes, the other nodes started to adapt to his rhythm. Figure 7 shows how the nodes in the esperiment adjusted within just a few seconds.

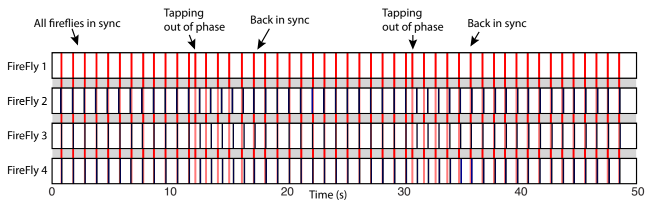


Figure 7. The figure shows the interaction between four nodes when one of them is controlled using the wearable sensor. Each vertical line denotes the firing of a firefly. The firings of fireflies 2, 3 and 4 are in black and the firings of the first firefly are marked in red extended across the entire range of fireflies to facilitate assessment of the degree of synchrony. The phase of node 1 is overridden twice by tapping while wearing the sensor shoe. This is deliberately done out of phase. Notice how nodes 2-4 adjust to the phase of the first node within a few seconds. In this example $\alpha = 0.1$.

IV. The Challenge: Frequency Adjustment in Pulse-Coupled Oscillators

The approach presented above assumes the frequencies of all the nodes to be fixed and equal. When this assumption is not made, the problem becomes more relevant to a real musical application, since human users may find it difficult to keep a steady beat. At the same time, a system with different frequencies is also much more complex. Consequently, the approach from phase synchronisation is not directly applicable in frequency synchronisation. We are working on this challenge and would like to propose some guidelines that may act as a roadmap towards solving frequency synchronisation in decentralised interactive music systems. Significant differences in starting frequency can be allowed if certain considerations are made regarding the synchronisation objective and the function used to update the frequency of each node, as will be discussed below.

A. Harmonic Synchrony

Some publications (e.g. [15]) using pulse-coupled oscillators allow the nodes to have slightly different frequencies, since minor differences in frequency will be overridden by phase adjustments. However, in our musical system, we would prefer the possibility of having a large deviation in starting frequencies. When listening to music people do not necessarily agree on a common pulse; while some people may entrain to one tempo, others might find the double or half of this tempo to be more natural [17]. This is a good reason for modifying the synchronisation goal in interactive music systems.

To illustrate the challenge, Figure 8 shows three oscillators with large differences in frequency. In the figure, the frequencies of node 2 and 3, respectively, are close to half and double that of node 1. A normal synchronisation objective would be for all of the nodes to fire at the same time. Such an objective, here called *strict synchrony*, would require all of the nodes to converge toward the same frequency. In cases like the one in Figure 8, it might be a more suitable approach to allow the frequencies of nodes to be integer multiples of the node with the lowest frequency. This would allow nodes to obtain a state of what we call *harmonic synchrony*.

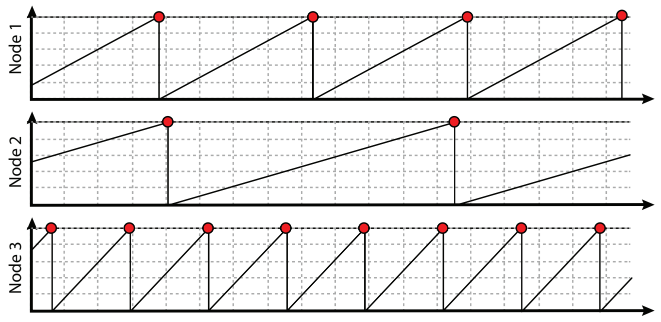


Figure 8. Three nodes with large differences in frequency. In this case, harmonic synchrony may be a more suitable objective than strict synchrony.

B. Bi-directional frequency adjustment

The phase update function presented in Section III causes positive phase jumps when a fire event is received. The use of only positive phase jumps is possible because ϕ resets to 0 as soon as it reaches 1. The frequency update function cannot rely solely on positive phase jumps, since this would cause nodes to increase the frequency uncontrollably and eventually reach an unstable state. For this reason, the frequency update function must be able to cause both positive and negative adjustment of the frequency. We suggest using a function that is positive when $\phi < 0.5$ and negative when $\phi > 0.5$. Or in other words, if a node is less than half-way through its cycle when a fire event is received, it should increase its frequency to "catch up" with the firing node. For example, by using the factor ρ calculated as follows:

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

C. Self-awareness

To strengthen the nodes in the system that are the most synchronised with the rest of the group, a well-synchronised node should make less adjustments to its own frequency compared with a poorly synchronised node. For this to be possible, a node should be *self-aware*, here understood as being able to assess its own level of synchrony with the other nodes.

We suggest the use of an error-measure for a node to assess its own level of synchrony. Each time a node detects a fire event from another node, it calculates a value, $\epsilon \in [0, 1]$, which is at its highest value when $\phi = 0.5$, and lowest value when ϕ is equal to 0 or 1. If we let

$$\epsilon = sin(\pi\phi(t))^2, \tag{4}$$

with the special case that $\epsilon = 0$ if a fire event is perceived within the refractory period, we can use $\epsilon(n)$ as a discrete function of the n-th fire event received by a node. Self-synchrony-assessment of node, s, may then be calculated by applying a running median filter to $\epsilon(n)$:

$$s = \text{median}\{\epsilon(n), \epsilon(n-1), ..., \epsilon(n-m)\}, \tag{5}$$

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 perceived fire events. The effect of using s as a factor in the frequency update function is that nodes with a high level of synchrony make smaller changes in frequency.

To update the frequency of a node, we specify the discrete function $H(n)$ for the n-th perceived fire event:

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

where $\rho(n)$ and $s(n)$ are discrete functions of the fire events perceived by a node. Note that H (as opposed to P from section III) does not output a new value for ω, but rather a value between -1 and 1 indicating whether ω should be decreased or increased.

D. The Reachback Firefly Algoritm

In section III, phase adjustment was done immediately whenever a node received a fire event from another node. With the suggested function, H, for frequency adjustment, immediate changes in frequency might potentially cause a bias towards increase in frequency, since the time period when $\phi > 0.5$ gets shorter if a fire event is received before ϕ reaches 0.5. To prevent this bias, we suggest a variation of the *reachback firefly algorithm* (RFA) [18]. 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. Figure 9 illustrates application of RFA in the frequency update function.

We may summarise the considerations provided above in the following frequency update function:

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

where $\beta \in [0, 1]$ is a constant denoting the coupling strength between the nodes, and y is the number of received fire events during the latest oscillator period. This function allows $\omega_i(t^+)$ values in the range $\frac{1}{2}\omega_i(t)$ to $2\omega_i(t)$. The extreme values occur only when $\phi = 0.25$ or 0.75 and $\beta = s = 1$.

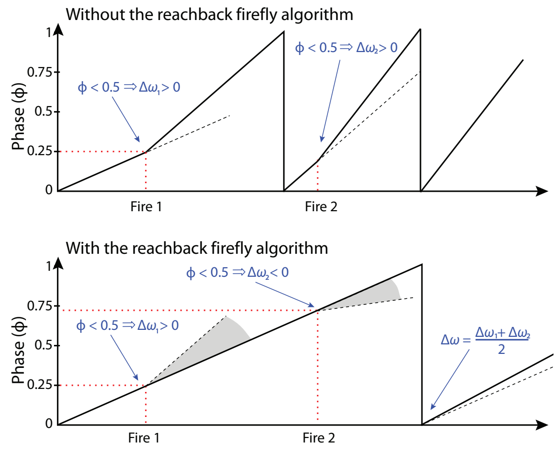


Figure 9. The top figure displays how the frequency adjustment of a node may be biased towards positive values when received fire events causes immediate frequency adjutments. The bottom figure shows how a variant of the reachback firefly algorithm may be used to tackle this problem. The algorithm collects values from the H function during its entire preiod, and adjusts the frequency at the end of the period by the mean of these values.

We believe these considerations may act as a roadmap towards solving the problem of frequency synchronisation in collaborative interactive music systems.

V. Conclusions and Future Work

We have presented an implementation of a variant of the Mirollo-Strogatz algorithm for phase-synchronisation of pulse-coupled oscillators running in iOS aimed at interactive and collaborative musical systems. In most of the tested cases, the implementation obtained a state of synchrony within 10 seconds, regardless of the number of nodes (up to 30 simultaneous nodes were tested). When introducing an out-of-sync human-controlled node to the group, the group quickly synchronised to the beat of the human-controlled node. Further, we have presented the challenge of decentralised frequency synchronisation in such systems, and a roadmap towards a potential solution. We suggest a redefinition of the synchrony objective, by allowing nodes to fire at frequencies that are integer multiples of other nodes, and to incorporate self-assessment within a node of the degree of synchrony with other nodes.

Acknowledgment

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

References

[1] Harmonix, “Guitar hero (software),” Red Octane, 2005.

[2] PG Music Inc., “Band in a Box (software),” 1990.

[3] B. Moens, L. van Noorden, and M. Leman, “D-jogger: Syncing music with walking,” in *Proc. of SMC 2010 7th Sound and Music Computing Conference*, 2010, pp. 451–456.

[4] G. Wang, “Designing smule’s ocarina: The iphone’s magic flute,” in *Proc. of Int. Conf. on New Interfaces for Musical Expression*, Pittsburgh, PA, 2009, pp. 303 – 307.

[5] A. Chandra, K. Nymoen, A. Voldsund, A. R. Jensenius, K. Glette, and J. Torresen, “Enabling participants to play rhythmic solos within a group via auctions,” in *Proceedings of the 9th International Symposium on Computer Music Modelling and Retrieval*, London, 2012, pp. 674–689.

[6] 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 *Int. Conf. Self-Adaptive and Self-Organizing Systems Workshops (SASOW)*. IEEE Comp. Society, 2011, pp. 102–107.

[7] I. Cross, “Music, cognition, culture, and evolution,” *Ann. N. Y. Acad. Sci.*, vol. 930, no. 1, pp. 28–42, 2001.

[8] R. E. Mirollo and S. H. Strogatz, “Synchronization of pulse-coupled biological oscillators,” *SIAM Journal on Applied Mathematics*, vol. 50, no. 6, pp. 1645–1662, 1990.

[9] A. T. Winfree, “Biological rhythms and the behavior of populations of coupled oscillators,” *Journal of Theoretical Biology*, vol. 16, no. 1, pp. 15 – 42, 1967.

[10] O. Babaoglu, T. Binci, M. Jelasity, and A. Montresor, “Firefly-inspired heartbeat synchronization in overlay networks,” in *Int. Conf. Self-Adaptive and Self-Organizing Systems, 2007. SASO ’07.*, 2007, pp. 77–86.

[11] R. Leidenfrost and W. Elmenreich, “Firefly clock synchronization in an 802.15.4 wireless network,” *EURASIP Journal on Embedded Systems*, vol. 2009, pp. 1–17, 2009.

[12] A. Christensen, R. O’Grady, and M. Dorigo, “From fireflies to fault-tolerant swarms of robots,” *Evolutionary Computation, IEEE Transactions on*, vol. 13, no. 4, pp. 754–766, 2009.

[13] J. Klinglmayr, C. Kirst, C. Bettstetter, and M. Timme, “Guraranteeing global synchronization in networks with stochastic interactions,” *New Journal of Physics*, vol. 14, 2012.

[14] Y. Kuramoto, “Collective synchronization of pulse-coupled oscillators and excitable units,” *Physica D: Nonlinear Phenomena*, vol. 50, no. 1, pp. 15–30, 1991.

[15] J. Klinglmayr and C. Bettstetter, “Self-organizing synchronization with inhibitory-coupled oscillators,” *ACM Transactions on Autonomous and Adaptive Systems*, 2012.

[16] D. Iglesia, “Mobmuplat (iOS application),” 2013.

[17] F. Styns, L. van Noorden, D. Moelants, and M. Leman, “Walking on music,” *Human Movement Science*, vol. 26, no. 5, pp. 769 – 785, 2007, music, Movement, Sound and Time, 10th Rhythm Perception and Production Workshop.

[18] 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 3rd international conference on Embedded networked sensor systems*, ser. SenSys ’05. New York, NY, USA: ACM, 2005, pp. 142–153.