Chapter 4

Implementation

This chapter gives an overview of the developed musical multi-robot system, methods implemented for it, as well as the performance measure used to evaluate these methods. The main goal of the implemented system is to allow for individual musical agents in a musical multi-agent collective to interact with each other, in order to achieve emergent and co-ordinating behaviour—in our case synchronization—with varying degrees of self-awareness, collective-sizes, and of difficulty and certainty in the environment and communication. More specifically, the goal with the design is to enable the robot collective to achieve so-called *harmonic synchronization* within a relatively short time. Exactly what is meant by *harmonic synchronization* will be expounded in Section 4.4.

These goals firstly require of the agents the modelling of oscillators with their properties, like phase and frequency, as explained further in Subsection 4.1.1. To allow for interaction and communication between the agents, mechanisms so that the agents can transmit "fire"-signals, as well as listen for other agents's "fire"-signals, is necessary as well, and is presented in Subsection 4.1.2.

First, the system and the system components will be presented and introduced. Then, methods implemented for achieving the system target goal of *harmonic synchrony* in various synchronization objectives—firstly solely for oscillator-phases, then secondly for both oscillator-phases and oscillator-frequencies—will be described and presented.

A helpful way to read this chapter could be as answers to the question "how can one achieve and detect the final part of the chapter by performing the middle part to the first one?" — more specifically "how can one achieve and detect the target/goal state of harmonic synchrony, by performing updates to the musical robots's phases and frequencies, in the musical multi-robot collective?"

4.1 The musical multi-robot collective

Envision that we have a decentralized (i.e. no central control) multi-agent collective scenario consisting of musical robots modelled as oscillators. These are solely communicating through brief "fire"-like audio-signals—greatly inspired by K. Nymoen et al.'s synchronizing "fireflies" [3]. They are not initially synchronized in their firing of audio-signals; but as time goes, they are entraining to synchronize to each other by adjusting their phases and frequencies when/after

hearing each other's audio-/fire-signals. If they then, after some time of listening and adjusting, succeed in becoming synchronized — we then will eventually see "fire"-events/-signals line up at an even underlying pulse or rhythm. Examples and demonstrations of this process are depicted in Figure 4.1 and Figure 4.2.

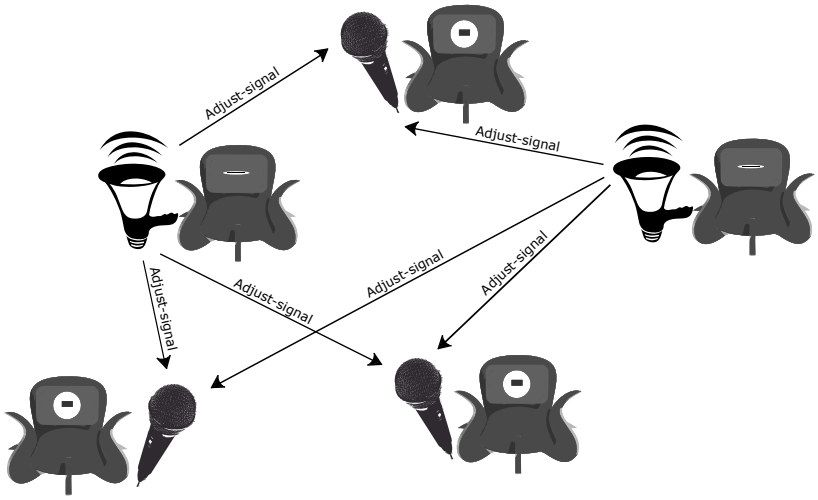
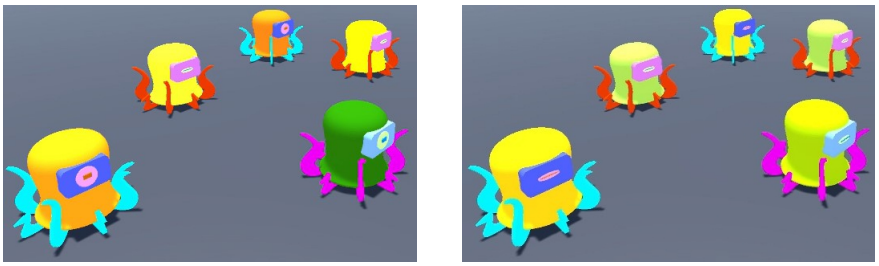


Figure 4.1: Illustration/Schematic: Musical robot collective entraining to synchronize to each other, or more specifically achieve harmonic synchronization, through performing phase- & frequency-adjustments. Agents that are not firing at the moment will adjust themselves after hearing a transmitted "fire"-/adjustment-signal from a neighbouring firing agent.



(a) Agents firing, i.e. blinking with their eyes and turning their body yellow, asynchronously at first. Only two Dr. Squiggles-robots with red tentacles fire simultaneously, the rest do not.

(b) Seconds later, after having listened to each others's adjustment-signals and adjusted themselves accordingly, all agents now fire simultaneously and synchronously.

Figure 4.2: From simulation: Example of harmonic synchronization achieved in a musical robot collective, where frequencies are equal and constant, but phases are initially heterogenous and a uniformly random number $\in [0, 1]$.

4.1.1 The individual agent: a musical robot

The musical robot collective consists of M. J. Krzyzaniak and RITMO's musical robots, the Dr. Squiggles [4]. The 3D-models of these robots were designed by Pierre Potel [5] and were reused, with permission, for the simulations in this thesis project.

Every musical robot or node have certain components, attributes, and characteristics that make it what it is. Such include an oscillator-component, consisting of the agent's oscillator-phase ϕ and oscillator-frequency ω. Notions like "agent", "robot", "firefly", and "oscillator" will be used interchangeably throughout the thesis. The agents have an input-mechanism for hearing/detecting transmitted "fire"-event signals from other agents, as well as an output-mechanism for transmitting or playing such "fire"-/adjust-signals or tones, as is illustrated with the microphone and megaphone respectively in Figure 4.1.

In order to be able to analyse the musical scenario within which they are situated (self-assessment), as well as for adapting their musical output accordingly (self-adaptation), the agents are to some extent endowed with artificial intelligence and self-awareness capabilities. The robots are self-aware of their own phase and frequency, but are unaware of other agents's true phases and frequencies. They also possess the self-assessment capability of evaluating how much in- or out-of-synch they are, as seen in the greater context of the entire robot collective. When the agents hear the transmitted "fire"-/adjust-signals, the agents are intelligent enough to adjust themselves in the direction of the system goal/target state.

4.1.2 Robot communication: the "fire"-signal

These aforementioned audio-signals, also referred to as "fire"-signals, "flash"-signals, or adjust-signals, are transmitted whenever an agent's oscillator *peaks* or *climaxes* (i.e. after its cycle or period is finished, having phase $\phi(t) = 1$) — or actually after every second *peak* due to the system target goal of *harmonic synchrony*, to be elaborated upon in Section 4.4.

The "fire"-signals are short and impulsive tones that the agents output through their loudspeakers. These short audio-signals/sounds "wildly" transmitted or played into the environment are then the only means of communication within the multi-agent collective, implying that are agents are pulse-coupled, not phase-coupled, oscillators. In other words, our agents will communicate and coordinate with each other through the very typical multi-agent system concept of *stigmergy*, meaning indirect communication and co-ordination by leaving traces of oneself in ones environment for others to observe or detect subsequently.

When an agent detects a "fire"-/adjust-signal, the agent will adjust its own oscillator-properties (phase ϕ and frequency ω), depending on which type of problem the agents are to solve. No individual agent is directly able to adjust or modify the state or properties of any other agent, only its own. Exactly the type of problems we attempt to solve in this thesis will be presented now in Section 4.2 and Section 4.3.

4.2 Synchronizing oscillator-phases

If we first assume constant and equal oscillator-frequencies in our agents, we can take a look at how the agents adjust their—initially random—phases in order to synchronize to each other. We will from here on and out refer to this first problem as **the ϕ-problem**, given that the phases (ϕ_i) for all agents i are what we need to adjust and synchronize — and that frequencies (ω_i) technically already are synchronized.

The goal state of the agents is now for all agents to fire/flash simultaneously, after having started firing/flashing at random initially. Note that this is a special case of the final and ultimate goal of *harmonic synchrony*. This is due to how all agents in the collective firing/flashing simultaneously, is considered having achieved harmonic synchrony since its phases would be synchronized if fire-events are lined up in even pulses, as well as all frequencies in the agent collective being within the set of "legal" frequencies, $\omega_0 \cdot 2^{\mathbb{N}_0}$, where ω_0 is the fundamental (smallest) frequency in the agent collective, and $0 \in \mathbb{N}_0$ — leading to $\omega_0 \cdot 2^0 = \omega_0$ to be a legal frequency, which is what all agents in our case here have as frequences.

In order for the musical agents to synchronize to each other, they will have to—due to their heterogenous and randomly initialized phases—adjust or update their own phases according to some well-designed update-/adjustment-functions, as presented below.

When it comes to the temporality and timing of when these updating functions are used and applied; Musical agents's phases get updated/adjusted immediately as "fire"-/"flash"-events from neighbouring robots are perceived.

4.2.1 Mirollo-Strogatz's "standard" phase-adjustment

One approach having been used to achieve this in the past is Mirollo-Strogatz's "Standard" Phase-adjustment in oscillators [2], as illustrated in Figure 4.3.

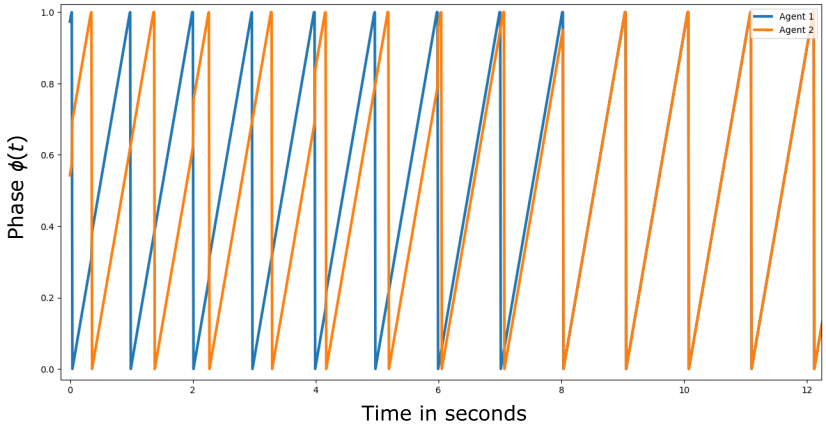


Figure 4.3: "Standard" phase-adjustment with Mirrollo-Strogatz's approach

Each musical agent gets a new phase, $\phi(t^+) = P(\phi(t))$, accoring to the **phase update function** (4.1) upon perceiving a "fire"-event from one of the other musical nodes:

$$\phi(t^+) = P(\phi(t)) = (1 + \alpha)\phi(t), \tag{4.1}$$

where "α *is the pulse coupling constant, denoting the strength between nodes*" [3], t^+ denotes the time-step immediately after a "fire"-event is heard, and $\phi(t)$ is the old frequency of the agent at time t. So, if e.g. $\alpha = 0.1$, then a musical agent's new and updated phase, immediately after hearing a "fire"-signal from another agent, will be equal to $\phi(t^+) = P(\phi(t)) = (1 + 0.1)\phi(t) = 1.1\phi(t)$. 110% of its old phase $\phi(t)$, that is. Hence, and in this way, the agent would be "pushed" to fire sooner than it would otherwise (as nodes fire once they have reached phase-climax $\phi(t) = 1$).

4.2.2 K. Nymoen's bi-directional phase-adjustment

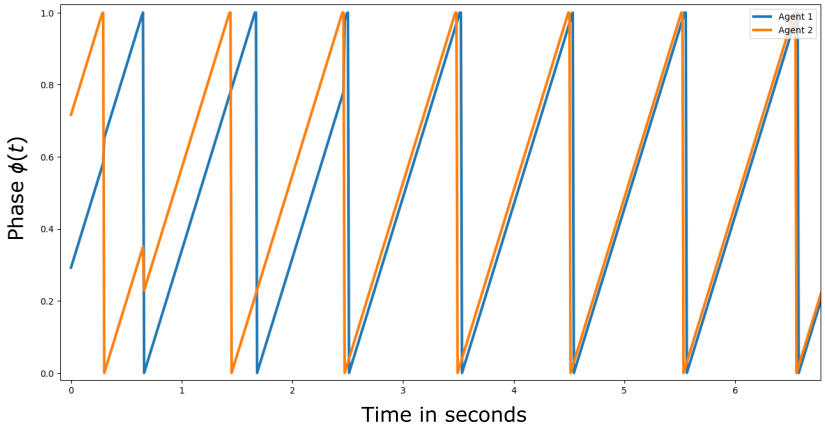


Figure 4.4: Bi-directional phase-adjustment with K. Nymoen et al.'s approach

This approach to phase-adjustment, as in Figure 4.4, works very similarly to the phase-adjustment performed in the "standard" ***Mirollo-Strogatz*** approach presented earlier; the only difference being that now, nodes update their phases with the slightly more complex **phase update function** (4.2) when hearing a "fire"-event from one of the other musical nodes — allowing for both larger, but also smaller, updated phases compared to the old phases:

$$\phi(t^+) = P(\phi(t)) = \phi(t) - \alpha \cdot sin(2\pi\phi(t)) \cdot |sin(2\pi\phi(t))| \tag{4.2}$$

, where t^+ denotes the time-step immediately after a "fire"-event is heard, and $\phi(t)$ is the old frequency of the agent at time t.

The fact that new and updated phases can both be larger, but also smaller, compared to the old phases, is exactly what's meant by the phase-adjustment being ***bi-directional***, or as the authors call it in the paper as using "*both excitatory and inhibitory phase couplings between oscillators*" [3].

The effects then of adjusting phases—upon hearing "fire"-events, according to this newest update-function (4.2)—are that the nodes's updated phases $\phi(t^+)$, compared to their old phases $\phi(t)$, now get decreased if $\phi(t)$ is lower than 0.5, increased if $\phi(t)$ is higher than 0.5, and neither—at least almost—if the phases

are close to 0.5. This is due to the negative and positive sign of the sinewave-component in Equation (4.2), as well as the last attenuating factor in it of $|sin(2\pi\phi)| \approx |sin(2\pi\frac{1}{2})| = |sin(\pi)| = |0| = 0$, then if we have $\phi(t) \approx 0.5 = \frac{1}{2}$.

4.2.3 Thorvaldsen's self-aware phase-adjustment

GJØR: [Beskriv min nye proposede algoritme (to-be-implemented) for å oppnå harmonisk synkronitet i ϕ-problemet med phase-adjustment, som inneholder flere tilleggs- Self-Awareness-komponenter, sammenliknet med K. Nymoens bi-directional phase-adjustment metode.

Eksempler på slike tilleggs- Self-Awareness-komponenter er såkalt Belief-awareness (som fanger usikkerhet og tillits-nivåer) og/eller Expectation-awareness (som kombinerer Belief-awareness og Time-awareness) [1]. Andre forslag er å implementere Self-Awareness i forhold til:

- avstand: f.eks. $H^*(n) = H(n) \cdot \frac{1}{distance}$ for "fire"-event n, så lenge ikke $distance = 0$.
- hvem som er hvem (i.e. agent_id).

].

4.3 Synchronizing oscillator-frequencies

When we open up for the possibility for heterogenous frequencies in our musical agent collective, we open up to exciting musical aspects like the playing of diverse rhythmic patterns as e.g. mentioned in Section 4.4, but we then also need to not only synchronize phases, but also frequencies, simultaneously. This second problem of adjusting synchronizing both all phases ϕ_i and frequencies ω_i, the values of which can all be heterogenous and initially random, for all agents i in the agent collective — we will from here on and out refer to as **the ϕ-&ω-problem**. This slightly more complex problem calls for us to also find solutions on how to adjust and synchronize frequencies. We already have methods by which we can adjust and synchronize the agents's phases ϕ with, described in Section 4.2, but we do so-far lack methods by which we can adjust and synchronize the agents's frequencies ω with.

We hence now introduce randomly initialized, non-constant, and heterogenous oscillator-frequencies in our musical agents. The agents are now required to synchronize their initially different and random frequencies, so that frequencies are "legal" and *harmonically synchronized*. Such "legal" frequencies are described clearly in detail in Section 4.4.

Three implemented approaches for achieving this are presented now. Notice the increasing degree of *Computational Self-Awareness* endowed in the methods.

4.3.1 Person X's low SA-leveled frequency-adjustment

GJØR: [Beskriv en simplere strategi/metode (to-be-implemented) for å oppnå harmonisk synkronitet i ϕ- & ω-problemet (i.e. problemet der både faser og

frekvenser starter med ulike og tilfeldige verdier, og altså begge trenger synkronisering) med frequency-adjustment, da uten noe Self-Awareness-egenskaper, hvis det finnes].

4.3.2 K. Nymoen's middle SA-leveled frequency-adjustment

This approach to Frequency Adjustment stands in contrast to previous approaches to synchronization in oscillators [fixed_freqs, fixed_range_freqs] where the oscillators's frequencies are either equal and fixed, or where frequencies are bound to initialize and stay within a fixed interval/range.

In order to achieve this goal of *harmonic synchrony* in conjunction with—or rather through—frequency adjustment, we have to go through a few steps to build a sophisticated enough update-function able to help us achieve this.

When it comes to the temporality and timing of when these update functions are used and applied; Musical agents's phases get updated/adjusted immediately as "fire"-/"flash"-events are perceived, whereas agents's frequencies do not get updated until the end of their oscillator-cycle (i.e. when having a phase-climax $\phi(t) = 1$). This is also the reason why frequencies are updated discretely, not continuously. So-called H-values however, being "contributions" with which the frequencies are to be updated according to, are immediately calculated and accumulated when agents are perceiving a "fire"-/"flash"-event — and then finally used for frequency-adjustment/-updating at phase-climaxes.

Each agent i update their frequency, on their own phase-climax (i.e. when $\phi_i(t) = 1$), according to the frequency-update function $\omega_i(t^+)$:

$$\omega_i(t^+) = \omega_i(t) \cdot 2^{F(n)}, \tag{4.3}$$

where t^+ denotes the time-step immediately after phase-climax, $\omega_i(t)$ is the old frequency of the agent at time t, and $F(n) \in [-1, 1]$ is a quantity denoting how much and in which direction an agent should update its frequency after having received its nth "fire"-signal.

This is how we obtain the aforementioned $F(n)$-quantity:

4.3.2.1 Step 1: the "in/out-of synch" error-measurement/-score, $\epsilon(\phi(t))$

Describing the error measurements at the n-th "fire"-event, we introduce an Error Measurement function.

The Error Measurement function (4.4), plotted in Figure 4.5, is calculated immediately by each agent i, having phase $\phi_i(t)$, when a "fire"-event signal from another agent is detected by agent i at time t.

$$\epsilon(\phi_i(t)) = sin^2(\pi\phi_i(t)) \tag{4.4}$$

As we can see from this Error-Function, the error-score is close to 0 when the agent's phase $\phi_i(t)$ is itself close to 0 or 1 (i.e. the agent either just fired/flashed, or is about to fire/flash very soon). This implies that if it was only short time

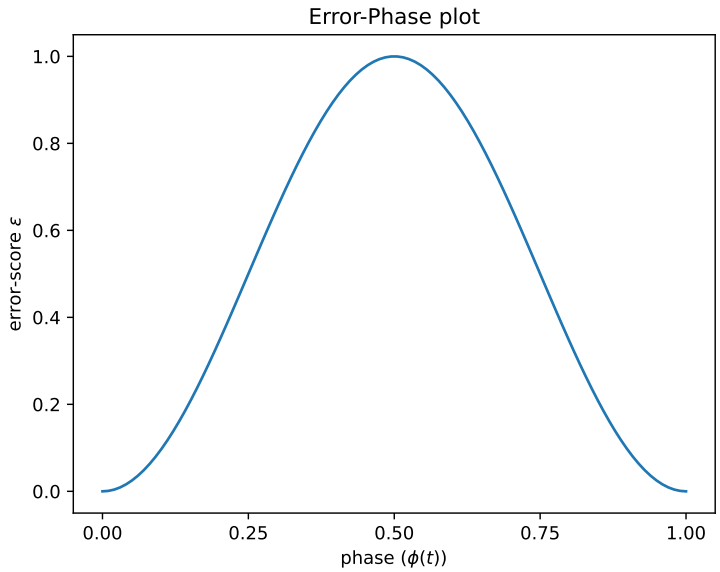


Figure 4.5: Error Measurement (4.4) plotted as a function of Phase

ago since we just fired, or conversely if there is only short time left until we will fire, we are not much in error or *out-of-synch*.

The error-score is the largest when an agent perceives a "fire"-signal while being half-way through its own phase (i.e. having phase $\phi(t) = 0.5$). We could also then ask ourselves, does not this go against the main/target goal of the system, being *harmonic synchrony* — if agents are allowed to be "half as fast" as each other? We could imagine a completely "legal" and harmonically synchronous scenario where two agents have half and double the frequency of each other. The agent with half the frequency of the faster agent would then have phase $\phi(t) = 0.5$ when it would hear the faster agent "fire"/"flash" — leading to its Error-score $\epsilon(0.5) = sin^2(\pi/2) = 1$, which then makes it seem like the slower agent is maximally out of synch, when it is actually perfectly and harmonically synchronized. This calls out for an attenuating mechanism in our frequency update function, in order to "cancel out" this contribution so that perfectly harmonically synchronized agents will not be adjusted further despite their high Error-measurement. As we will see below, in Figure 4.6, exactly such an attenuating mechanism is utilized in our frequency-adjustment method.

This Error-Measurement/-Score forms the basis and fundament for the first component of Self-awareness, being the *self-assessed synchrony-score* $s(n)$.

4.3.2.2 Step 2: The first self-awareness component, s(n)

This aforementioned self-assessed synchrony-score, $s(n)$, is in fact simply the median of a list containing m Error-scores ϵ. Such a list is easily implemented in C# by declaring a List<float> called *errorBuffer* e.g. (i.e. *errorBuffer* is a list containing floating point values):

$$errorBuffer = \{\epsilon(n), \epsilon(n-1), ..., \epsilon(n-m)\}, \tag{4.5}$$

then leading to:

$$\begin{aligned} s(n) &= median(errorBuffer) \\ &= median(\{\epsilon(n), \epsilon(n-1), ..., \epsilon(n-m)\}) \in [0,1], \end{aligned} \tag{4.6}$$

where n is the latest observed "fire-event", and m is the number of the last observed "fire"-events we would like to take into account when calculating the self-assessed synch-score.

If we then have a high $s(n)$-score, it tells us that the median of the k last error-scores is high, or in other words that we have mainly high error-scores — indicating that this agent is out of synch. Conversely, if we have a low $s(n)$-score, indicating mainly low error-scores for the agent — then we have an indication that the agent is in synch, hence leading to low error scores, and in turn low $s(n)$-scores.

In other words, each agent hence has a way to assess themselves in how much in- or out-of-synch they believe they are compared to the rest of the agents. This is then the first degree/aspect of public[?] self-awareness in the design.

4.3.2.3 Step 3: frequency update amplitude- & sign-factor, $\rho(n)$

Describing the amplitude and sign of the frequency-modification of the n-th "fire-event" received. It is used to say something about in which direction, and in how much, the frequency should be adjusted.

$$\rho(\phi) = -sin(2\pi\phi(t)) \in [-1,1] \tag{4.7}$$

For example, if an agent i has phase $\phi_i(t) = 1/4$, it gets a value $\rho(1/4) = -sin(\pi/2) = -1$; meaning, the agent's frequency should be decreased (with the highest amplitude actually) in order to "slow down" to wait for the other nodes. Conversely, if an agent j has phase $\phi_j(t) = 3/4$, it gets a value $\rho(3/4) = -sin(3/2\pi) = -(-1) = 1$; meaning, the agent's frequency should be increased (with the highest amplitude) in order to getting "pushed forward" to catch up with the other nodes.

Acts as an attenuating factor, when $\phi(t) \approx 0.5$, in the making of the H-value — supporting the goal of *harmonic synchrony*.

4.3.2.4 Step 4: the H-value, and the H(n)-list

The following value, being "frequency-update-contributions", is then (as previously mentioned) calculated immediately when the agent perceives another agent's "flashing"-signal:

$$H(n) = \rho(n) \cdot s(n) \tag{4.8}$$

Here we then multiply the factor $\rho(n)$, depicted in Figure 4.6, representing how much, as well as in which direction, the agent should adjust its frequency, together with a factor $s(n) \in [0,1]$ of the adjusting agent's self-assessed synch-score. This means that all the possible values this $H(n)$-value can take, lies within the green zone in Figure 4.7. We hence see that the smallest value $H(n)$

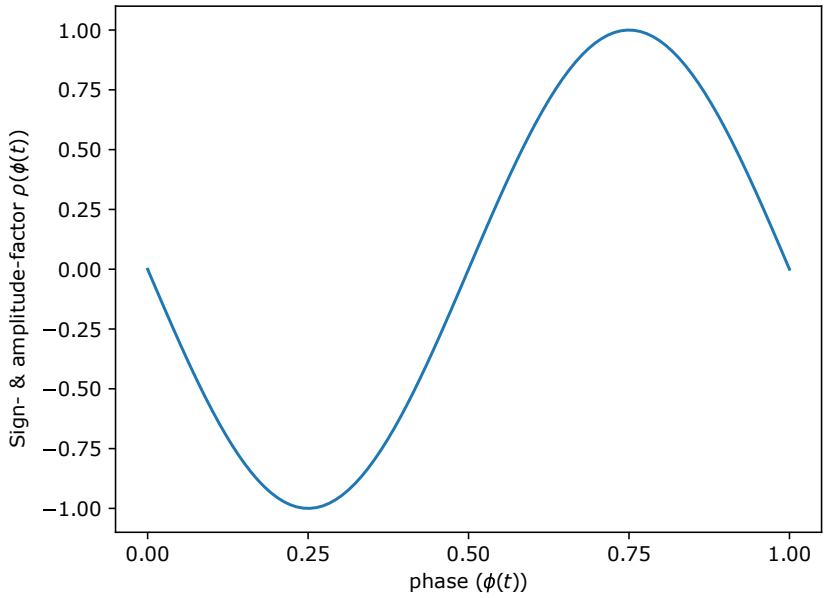


Figure 4.6: The amplitude- & sign-factor, $\rho(\phi(t))$, where $\phi(t)$ is the phase of an agent at time t when it heard "fire"-event n. Notice how the amplitude is equal to 0 when the phase is equal to 0,5.

can take for the nth "fire"-event is -1, which it does when $\phi(n) = 0.25$ and $s(n) = 1$. The highest value it can take is 1, which it does when $\phi(n) = 0.75$ and $s(n) = 1$. We can also see that even though the self-assessed synch-score $s(n)$ (i.e. the median of error-scores) is high and even the maximum value of 1, thus indicating consistent high error-scores (judging by error-function (4.4) and Figure 4.5) — the "frequency-update-contribution" $H(n)$ can in the end be cancelled out, as alluded to before, if in fact the amplitude- & sign-factor $\rho(n)$ is equal to 0. Hence, if we have two agents then where the one is twice as fast as the other, and we accept the $H(n)$-value as the "frequency-update-contribution", the slower agent which will hear "fire"-events consistently when it has its phase $\phi(n) \approx 0.5$ (if the agents are synchronized) will, even though it gets a high *out-of-synch score* $s(n) \approx 1$, not "be told" to adjust its frequency more by getting a large "frequency-update-contribution", but in fact "be told" not to adjust its frequency more due to the small or cancelled-out "frequency-update-contribution."

To recall, the self-assessed synch-score $s(n)$ tells an adjusting agent how in- or out-of-synch it was during the last m perceived "fire"-/"flash"-events — where $s(n) = 0$ signifies a mean of 0 in error-scores, and $s(n) = 1$ signifies a mean of 1 in error-scores. So then if this H-value is to be used to adjust the nodes's frequencies with, the frequency will then be adjusted in a certain direction and amount (specified by $\rho(n)$) — given that the agent is *enough* "out of synch"/"unsynchronized" (in the case $s(n)$ is considerably larger than 0).

The H-value says something about how much "out of phase" the agent was at the time the agent's nth "flashing"-signal was perceived (and then followingly how much it should be adjusted, as well as in which direction after having been

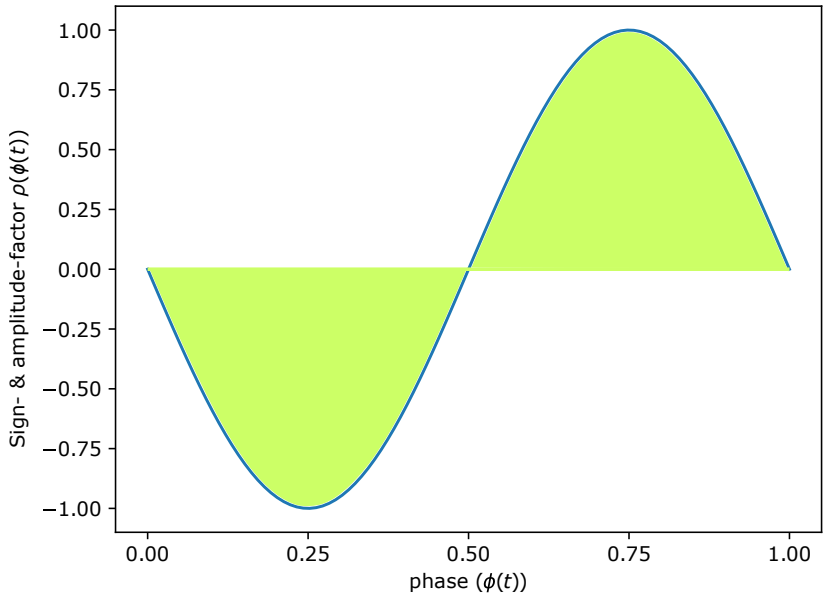


Figure 4.7: All possible $H(n)$-values marked in green, given by $\rho(n) \cdot s(n)$, where $\rho(n)$ is defined as above and depicted in Figure 4.6, and the self-assessed synch-scores $s(n) \in [0, 1]$. The $s(n)$-scores were again simply the median of m error-scores $\{\epsilon(n), \epsilon(n-1), \epsilon(n-2), \ldots, \epsilon(n-m)\}$ which again were given by the error-function (4.4) plotted in Figure 4.5.

multiplied together with a sign-factor $\rho(n)$), given then that this H-value also consists of the *self-assessed synch score s(n)* — which again simply was the median of the list of error-scores, *errorBuffer* in our case.

We could look at this H-value as representing the direction and amplitude of the frequency adjustment weighted by the need to adjust (due to being out of synch) at the time of hearing "fire"-/"flash"-event n. Or in other words, this H-value is then the n-th contribution with which we want to adjust our frequency with.

Especially interesting cases are when we have $\phi(n) \approx 0.5 \implies \rho(n) \approx \pm 0$, as well as the last m Error-scores $\epsilon(n)$ being close to 0, also leading to $s(n) \approx 0$. In both of these two cases the entire frequency-adjustment contribution H would be cancelled out, due to harmonic synchronization (legally hearing a "fire"-/"flash"-event half-way through ones own phase) in the first case, and due to not being out of synch in the latter (having low Error-Measurements). Cancelling out the frequency adjustment contribution in these cases is then not something bad, but something wanted and something that makes sense. If these H-values then are cancelled out or very small, it is indicative of that nodes are already in *harmonic synchrony*, and hence should not be "adjusted away" from this goal state. On the other side, if these H-values then are different (e.g. closer to -1 and 1), it is indicative of that nodes are not yet in *harmonic synchrony*, and that they hence should be "adjusted closer" to the goal state.

All the calculated H-values are in my implementation accumulated and stored in an initially empty C#-list (of floats), referred to as $H(n)$, at once they

are calculated. The $H(n)$-list is then consecutively "cleared out" or "flushed" when its H-values have been used for the current cycle/period's frequency adjustment (i.e. at the phase climax, when $\phi(t) = 1$), and is then ready to accumulate new H-values during the next cycle/period.

4.3.2.5 The final step: the frequency update function, $\omega_i(t^+)$

Now, we can pull it all together, for Nymoen et al.'s Frequency Adjustment approach for achieving harmonic synchrony with initially randomized and heterogenous frequencies.

When an agent i has a phase-climax ($\phi_i(t) = 1$), it will update/adjust its frequency to the new $\omega_i(t^+)$ accordingly:

$$\omega_i(t^+) = \omega_i(t) \cdot 2^{F(n)}, \tag{4.9}$$

where t^+ denotes the time-step immediately after phase-climax, and $F(n)$ is found by:

$$F(n) = \beta \sum_{x=0}^{k-1} \frac{H(n-k)}{k}, \tag{4.10}$$

where β is the frequency coupling constant, k is the number of heard/received "fire-event"s from the start of the last cycle/period to the end (i.e. the phase-climax, or *now*) — and the rest of the values are as described above.

This $F(n)$-value then, as we see in Equation (4.10), is a weighted average of all the agent's $H(n)$-values accumulated throughout the agent's last cycle.

4.3.3 Thorvaldsen's high SA-leveled frequency-adjustment

GJØR: [Beskriv min nye proposede algoritme (to-be-implemented) for å oppnå harmonisk synkronitet i ϕ- & ω-problemet med frequency-adjustment, som inneholder flere tilleggs- Self-Awareness-komponenter, sammenliknet med K. Nymoens frequency-adjustment metode. Eksempler på slike tilleggs- Self-Awareness-komponenter er såkalt Belief-awareness (som fanger usikkerhet og tillits-nivåer) og/eller Expectation-awareness (som kombinerer Belief-awareness og Time-awareness) [1]. Andre forslag er å implementere Self-Awareness i forhold til, og vekte frekvensoppdaterings-bidragene H(n) i henhold til:

- avstand: f.eks. $H^*(n) = H(n) \cdot \frac{1}{distance}$ for "fire"-event n, så lenge ikke $distance = 0$.
- hvem som er hvem (i.e. agent_id).
- større median-liste ift. den self-assessed'e synch-score'n $s(n)$ og $errorBuffer$'et. Dette kan være nyttig ved større collective-sizes, da et lite/kort median-filter/-$errorBuffer$-liste vil kunne miste eller gå glipp av error-scores, $\epsilon(n)$, fra "fire"-events fra langt tilbake (tidlig) i "oppsamlings-perioden."
- de andres frekvenser. Høre etter og registrere andre individers "fire"-signaler kontinuerlig og estimere disse individenes frekvenser utifra det (f.eks. $\hat{\omega}_j = time_{j,fired\_now} - time_{j,fired\_last\_time}$, eller et gjennomsnitt av slike oppsamlede verdier).

].

4.4 System target state: harmonic synchrony

As previously mentioned, this state of *harmonic synchrony* is then the system goal state which we want the musical robot-collective to end up in eventually, and the sooner the better.

The state of harmonic synchrony is defined as the state in which all agents in the musical collective "fire"/"flash", as described in Subsection 4.1.2, at an even and underlying interval or pulse, a certain number of times in a row. This is not to say all agents will have to "fire"/"flash" simultaneously, as has traditionally been the case for pulse-coupled oscillators []. So in a way, for phases ϕ to be harmonically synchronized, all agent "fire"-signals will have to coincide in a way, even though they don't all have to incur exactly at the same time always. Exactly how this can look is (to-be) shown in Section 4.5.

As one is designing and creating an interactive music technology system, one might want to encourage and allow for the playing of various musical instruments at various rhythms/paces, as it might be quite boring if all instruments were played at the exact same measure or pulse. As K. Nymoen et al. [3] reason when discussing their own interactive "Firefly" music-system, as well as coining the term of harmonic synchrony:

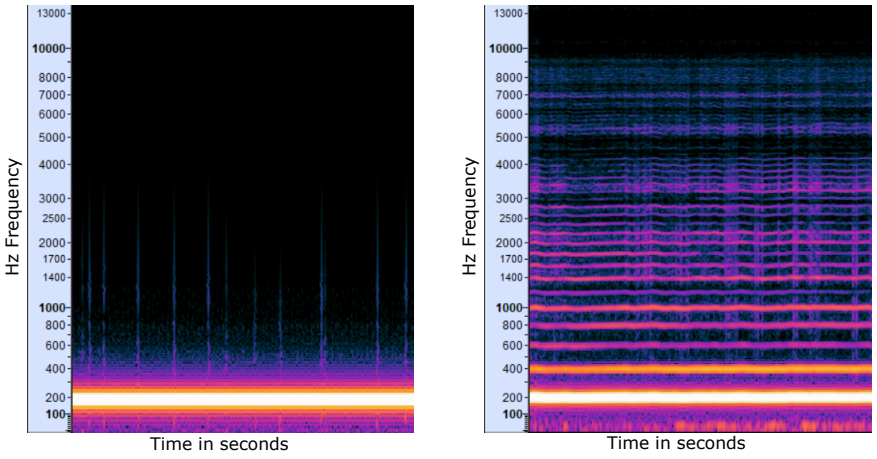
"*Temporal components in music tend to appear in an integer-ratio relation to each other (e.g., beats, measures, phrases, or quarter notes, 8ths, 16ths).*"

and

"*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).*"

Accomodating for these aspects then, K. Nymoen et al. [3] took inspiration for achieving synchronization in a decentralized system from the concept of *harmonics* in the frequency spectrum of a waveform, in that each harmonic wave or overtone has a frequency with an integer-relationship to the fundamental (smallest) frequency. This phenomenon can e.g. be seen in the frequency spectrogram of a humanly hummed G3-tone, in Figure 4.8b, where we can observe the harmonics and overtones having frequencies with integer relationships to the fundamental (smallest) frequency, which was intended to be 195,99 Hz.

More accurately then, all musical agents—in a harmonically synchronized state—will have frequencies $\in \omega_0 \cdot 2^{\mathbb{N}_0}$, where ω_0 is the agent with the lowest frequency's frequency (i.e. the fundamental frequency), and $\mathbb{N}_0$ is the mathematical set of natural numbers including the number zero. Hence, agents will typically have frequencies like $\omega_0 \cdot 2^0 = \omega_0$, $\omega_0 \cdot 2^1 = 2\omega_0$, $\omega_0 \cdot 2^2 = 4\omega_0$, or $\omega_0 \cdot 2^3 = 8\omega_0$. If all agents end up with these kind of frequencies, we say they have "legal" and harmonically synchronized frequencies.



(a) The frequency spectrogram of the audible waveform being a monotone and purely generated G3-tone at 195,99 Hz [6].

(b) The frequency spectrogram of the audible waveform being a more-or-less monotone but non-pure G3-tone, hummed and recorded by me [], as I tried to repeat the tone depticted in 4.8a with my voice.

Figure 4.8: Frequency spectrograms of two different-sounding waveforms of the same G3-tone at 195,99 Hz. Frequencies in a harmonically synchronized agent collective will for the first ϕ-problem resemble the frequencies in 4.8a, since all frequencies in this problem are equal and constant. Conversely, when frequencies can be heterogenous, as in the ϕ- & ω-problem, the frequencies in a harmonically synchronized agent collective will rather resemble the frequencies in 4.8b, where harmonics and overtones (i.e. frequencies in integer-relationships to the fundamental and lowest frequency) are present.

4.5 Performance-measure: time until system target state of harmonic synchrony is detected

The performance-measure we will use to evaluate our implemented methods with, is the time it takes for the agent collective to achieve the system target state of harmonic synchrony, also referred from here on and out as HSYNCHTIME. The question then becomes: how does one detect harmonic synchrony in an musical agent collective?

In order for the collective to be considered harmonically synchronized, a set of conditions have to be met:

- Firings 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.

E.g., as K. Nymoen et al. did, one could have the short time-period during which agents are allowed to fire, t_f, be equal to say 80ms, and $k = 8$. t_q is not defined, as it depends on the frequency to which the nodes converge.

t_q is also then reset when a firing is observed within the "illegal" t_q-window.

When all the conditions above have been met, we have detected harmonic synchrony, and we also then have our synchrony-/performance-score, being the time from the start of the simulation until now, HSYNCHTIME.

An illustration (jeg vet den er litt utydelig og simpel foreløpig) of such a detection of harmonic synchrony is given below in Figure 4.9.

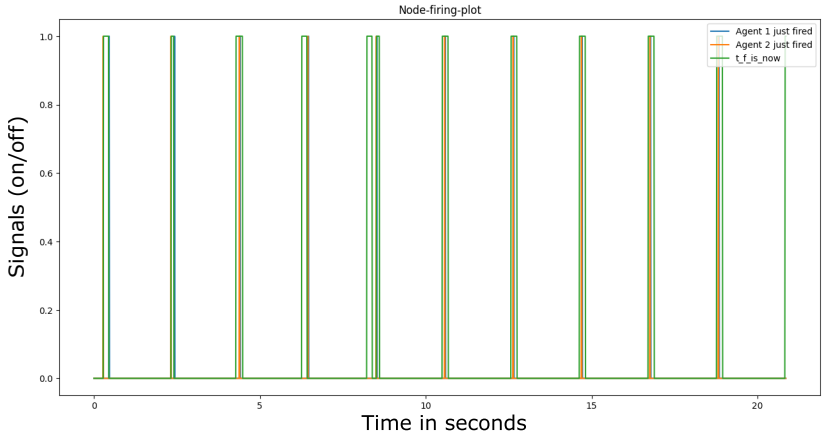


Figure 4.9: A node-firing plot, showing how the system target state of harmonic synchrony is detected eventually.

Bibliography

[1] Peter Lewis et al. "Towards a Framework for the Levels and Aspects of Self-aware Computing Systems". In: *Self-Aware Computing Systems*. Ed. by Samuel Kounev et al. Cham: Springer International Publishing, 2017. Chap. 3, pp. 51–85. ISBN: 978-3-319-47474-8. DOI: 10.1007/978-3-319-47474-8_3.

[2] Renato E. Mirollo and Steven H. Strogatz. "Synchronization of pulse-coupled biological oscillators". In: *SIAM Journal on Applied Mathematics* 50.6 (1990). Publisher: SIAM, pp. 1645–1662.

[3] Kristian Nymoen et al. "Decentralized harmonic synchronization in mobile music systems". In: *2014 IEEE 6th International Conference on Awareness Science and Technology (iCAST)*. IEEE, 2014, pp. 1–6.

[4] Silje Pileberg. *Say hi to the musical robot "Dr. Squiggles"*. URL: https://www.uio.no/ritmo/english/news-and-events/news/2021/drsquiggles.html (visited on 02/02/2022).

[5] Pierre Potel. *SoloJam-Island*. URL: https://github.com/67K-You/SoloJam-Island (visited on 02/02/2022).

[6] Tomasz P. Szynalski. *Online Tone Generator*. URL: https://www.szynalski.com/tone-generator/ (visited on 02/02/2022).