Computational Self-Awareness in Musical Robotic Systems

Endowing musical robots with self-awareness — an experimental and exploratory study

David Thorvaldsen

Thesis submitted for the degree of
Master in Informatics: Robotics and Intelligent Systems
60 credits

Institute for Informatics
Faculty of mathematics and natural sciences

UNIVERSITY OF OSLO

Spring 2022

Computational Self-Awareness in Musical Robotic Systems

Endowing musical robots with self-awareness — an experimental and exploratory study

David Thorvaldsen

Computational Self-Awareness in Musical Robotic Systems

http://www.duo.uio.no/

Printed: Reprosentralen, University of Oslo

Abstract

MSc Thesis/Project Summary.

Acknowledgements

Tusen takk til alle som har støttet og hjulpet meg. Sånn faktisk.

Soli Deo Gloria

Contents

List of Tables

List of Figures

latm

Chapter 1

Introduction

1.1 Motivation

Grunnene til å studere effektene av selv-bevissthet.

1.2 Goal of the thesis

BESKR.: ["to make the reader better understand what the thesis is about"—Jim, og "en rød tråd?"—Sigmund].

BESKR.: [Spesifikke mål (goals/aims) med master-oppgaven/-prosjektet. Hva jeg vil vise/demonstrere til folk (leserne f.eks.) om self-awareness (SA) og hvordan (detaljene av beskrevet i **Implementation**) — noe jeg håper å enten bekrefte eller avkrefte med eksperimenter og resultater i **Experiments and results**].

The specific aim of this thesis is to explore and investigate the effects of endowing robot systems with increased self-awareness abilities, and thus leads to the following research questions:

Research Question 1:
Will performance in collective multi-robot systems increase as the level of *self-awareness* increases? Specifically, will increased levels of self-awareness in the individual agents/musical robots lead to the collective of individuals being able to synchronize to each other faster than with lower levels of self-awareness?

Research Question 2:
Will increased levels of self-awareness lead to more robustness and flexibility in terms of handling environmental noise and other uncertainties — specifically in the continued ability of musical robots to synchronize to each other efficiently despite these difficulties/challenges?

1.3 Outline

En fin (Eagle's-eye) oversikt over strukturen til hele dette dokumentet fra nå av, og utover.

1.4 Scope and delimitations

BESKR.: [Hva oppgaven forsøker å oppnå eller besvare, og hva den ikke forsøker å oppnå eller besvare].

1.5 Contributions

BESKR.: [De viktigste bidragene av MSc thesis-arbeidet summert (for å få det til å stå frem/ut bedre enn å bare "gjemme" det i den siste delen av oppgaven). Fra Mats: HUSK Å SELG!].

This thesis work has led to some main novel contributions. The main contribution is that a novel synchronization-simulator has been created and developed in Unity, where various and customly made synchronization-methods can be implemented in a robot collective (either homogenously but also heterogenously in each robot); as well as that users of the simulator can both see and hear the robots's interactions and entrainment towards synchrony (including whether they manage to finally achieve synchrony, and in that case how long it takes them to get there). This novel framework opens, for everyone interested in it, for the possibility of experimenting with creative and novel synchronization-methods, in order to qualitatively and quantitatively assess their efficiency at achieving synchrony. A reason for this is due to the seamless scalability and dynamism, both in terms of robot-collective size, heterogenous or homogenous agents, as well as how a new update-function (for both phase-updating and frequency-updating) can so easily be added in and coded with a couple of lines.

Chapter 2

Background

BESKR.: [Surveying of past and related work].

BESKR.: [Clustering and classifications of main concepts and ideas used in the thesis].

2.1 Taking inspiration from psychology

GJØR: [Hent inn Essay-materiale inn hit].

GJØR: [Merge 1) Essay . Conceptual Framework med 2) "self awareness scope (i.e. individual robots are aware of and susceptible to a larger number of neighbouring robots and their transmitted "fire" signals)"].

2.2 Taking inspiration from natural phenomena

The intriguing, diverse, and complex phenomena of nature have for long served as exciting inspirations to human engineers and researchers [ant-colonies, boids, swarms, beeclust]. From taking inspiration when designing a robot traversing one of the most challenging terrains to traverse through, granular surfaces like sand that is, from e.g. zebra-tailed lizards able to run up to 5 meters per second (normally in desert sand) [1], to ... Such phenomena have been subject to considerable study and modelling—and have in fact created entire scientific fields [**bionics**, 3] in which principles from various science- & engineering-diciplines are applied to the physical systems and machines having functions that mimic biological processes, as well as biological processes serving as great sources of inspiration for the engineered systems; *Biomimetics* and *Bionics*, that is. One branch of these types of natural phenomena observed and studied is the biological pulse-coupled oscillators [**russerMinimalAssumptionsReferanser**].

An example of such biological pulse-coupled oscillators found in nature is the synchronously flashing fireflies, as e.g. can be seen in a dark forest in Figure 2.1.

Figure 2.1: Synchronous fireflies at Congaree National Park, United States. Photo: @columbiasc + @_flashnick (fotnote).

This phenomenon has inspired scientists like Mirollo & Strogatz [5] and in later time Kristian Nymoen et al. [6], to model and replicate this natural phenomenon in human-engineered systems. Given the periodic and repeating nature of the flashing/firing of the fireflies, modelling a firefly has been done by looking at each firefly as a periodic signal or oscillator. This work [5, 6] then ties into the broader work on synchronizing oscillators which has been subject to study for some time now []. What separates Mirollo-Strogatz and K. Nymoen's approaches from these other and previous oscillator-synchronizing methods, is mainly that here the oscillators are *pulse-coupled* (which the fireflies also can be said to be), as opposed to the more "standard" and constraining *phase-coupled* oscillators. Additionally, K. Nymoen's approach accounts for synchronizing initially heterogenous frequencies as well.

2.3 Signals and their responses

GJØR: [Forklare hva et signal eller et (lyd-)signal/waveform egentlig er (ift. harmonics/harmonic wave/overtones (og deres typiske integer-relationship frekvenser), og hvis tid timbre med varierende frekvenser (jf. Jay Eller hva han Andriy Burkov eller hva han het anbefalte), amplitude/gain, envelope/attack-decay)].

Signals, in various forms, are omnipresent in our world, whether we notice it or not. They are used for guidance: e.g. traffic-lights send visual light-signals to drivers to ensure traffic-flow and collision-avoidance; sirens and alarms fire away with the loudest of audio-signals (sounds) so that its listeners will get out of harms way; our nerves send pain-signals through our nervous-system if we touch something burning hot to ensure we do not damage our hand severely. If you have ever tried to make a scary sound to scare away an animal—like a cat or crow—they might in fact flee from you if they think your audible signal was scary enough for them.

A signal can also be called a waveform.

2.3.1 Amplitude

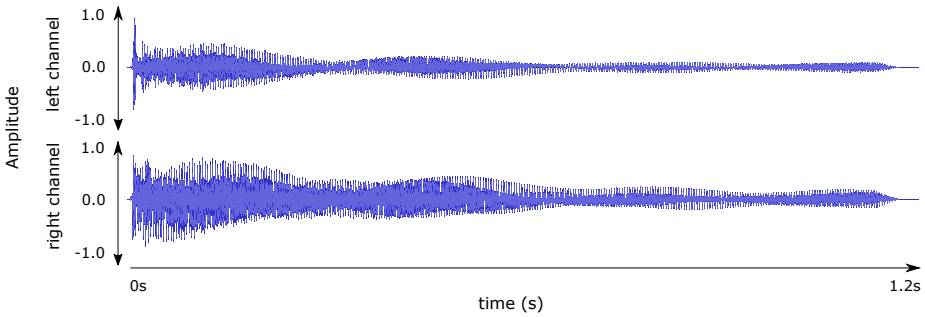


Figure 2.2: An audible stereo- (hence the two channels, one for each ear) waveform, being a digitally created harp-sound (as further described in 5.1.2). The more positive or negative the amplitude is, the louder the audio-signal is.

2.3.2 Frequency

2.3.2.1 Spectra / Spectrograms

2.4 Oscillators and oscillator-synchronization

GJØR: [Beskriv dette så godt at du kan snakke fritt om oscillatorers **faser** og **frekvenser** senere (i Implementation f.eks.), spesielt i tilfelle for noen ikke har vært borti det før, eller tatt et Signalbehandlings-kurs].

GJØR: [Skill på Pulse-coupled Oscillators, og Phase-coupled Oscillators. Nevn fordelene, spesielt for roboter, med å synkronisere seg basert på diskret tid [Sandsbots-paperet Kyrre sendte deg (iaff. slutt-tabellen), SandBots-arkitekturen som åpner for lave oppdaterings-rater], sammenliknet med kontinuerlig tid].

BESKR.: [Teori og nyere cutting-edge/state-of-the-art metoder for fase-/frekvens-synkronisering i oscillatorer og liknende. Se Kristian Nymoens referanser [6] og artikler funnet i denne veinen].

Oscillators have been used to implement and model a plethora of systems, also biological ones, ranging from designing the locomotion-patterns of swimming robot-amphibia through central pattern generators [2], ..., and as we have already established, modelling synchronously flashing or firing fireflies.

2.4.1 Phase and frequency

Much of the terminology from [6] is used here. An oscillator i is characterized by its *phase* $\phi_i(t)$, which is—at the start of its periodic signal period—initialized to

0 and evolves towards 1 at a rate which is called the *frequency* of the oscillator. So, in mathematical terms, the frequency $\omega_i(t)$ is then given by:

$$\omega_i(t) = \frac{d\phi_i(t)}{dt}. \tag{2.1}$$

When oscillator i's phase is equal to 1 (i.e. when $\phi_i(t) = 1$, or when the periodic signal period is over), we say oscillator i has *phase-climaxed*.

An oscillator-period T is defined as the inverse of the oscillator-frequency ω. In mathematical terms:

$$T = 1/\omega. \tag{2.2}$$

2.4.2 Phase-adjustment

GJØR: [Muligens inkluder introen i Section 5.2 her også (eller bare her?)].

BESKR.: [(Hvis relevante og ønskede) Tidligere metoder til Fase-synkronisering i oscillatorer (puls- og/eller fase-koplede)].

2.4.2.1 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 [5].

Each musical agent gets a new phase, $\phi(t^+) = P(\phi(t))$, accoring to the **phase update function** (2.3) upon perceiving a "fire"-event from one of the other musical nodes:

$$\phi(t^+) = P(\phi(t)) = (1 + \alpha)\phi(t), \tag{2.3}$$

where α is the pulse coupling constant, denoting the strength between nodes [6], 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 $\phi(t) = 1$).

2.4.3 Frequency-adjustment

GJØR: [Muligens inkluder introen i Section 5.3 her også (eller bare her?)].

BESKR.: [(If relevant and wanted) Previous approaches to Frequency-synchronization in oscillators (pulse- and/or phase-coupled) [nymoen-referanser e.g.] where the oscillators's frequencies are either equal and fixed, or where frequencies are bound to initialize and stay within a fixed interval/range.].

BESKR.: [(If it exists) A "simpler" frequency synchronization approach, to solve the phase and frequency (ϕ & ω) problem with, without any self awareness components].

2.5 Musical robots in music technology systems

GJØR: [Hent inn Essay-materiale om music technology systems her, spesielt om Solojam og kanskje Nymoen et al.s Firefly-system kjapt også (for å få en første idé før du går mer i detalj på dette systemet i **Baseline**)].

M. J. Krzyzaniak and RITMO's musical robots, the Dr. Squiggles, have been used in various music technology systems previously (fotnote) [7]. Corresponding 3D-models of these robots were designed by Pierre Potel [8] and are reused, with permission, for the simulation system designed during this thesis project.

Chapter 3

Baseline

BESKR.: [Mellomkapittel om K. Nymoens allerede-eksisterende approach og teori [6]. Her presenterer jeg teoriene og metodene jeg har brukt mesteparten av masterjobbings-tiden i starten på å implementere og teste i Unity].

Kristian Nymoen et al. [6] showed how one can, by amongst other measures endowing oscillators with self awareness capabilities, achieve synchronization (*harmonic* at that, cf. 3.3.1) of pulse coupled oscillators while synchronizing both phases and frequencies.

Other than endowing their firefly oscillators with self awareness capabilities, they developed update functions for both phases and frequencies which cause (close to) 0 change half-way through the oscillator cycle. These (close to) 0 changes half-way through robots's oscillator cycles enable individual musical robots to play e.g. half as quickly as other individuals—which typically characterizes musical interaction—as well as the possibility of achieving the target state of harmonic synchrony. Their contribution also differs from previous approaches in that Nymoen et al.'s fireflies only need to transmit "fire" signals every other phase climax, not every single phase climax.

The aforementioned update functions, as well as how additional self awareness capabilities are incorporated in one of them, are now presented in 3.1.1 and 3.2.1. Then, the relatively new system target state of *harmonic synchrony* will be expounded and explained in 3.3.

3.1 Phase synchronization

GJØR: [Introduser det første phase (ϕ) synchronization problemet, uthevet og i fet skrift. Deretter fortsett til løsningene dens (Phase-Adj.-metodene). "This is the first and simpler problem to solve, namely synchronizing the phases ϕ_i of all agents i."].

3.1.1 K. Nymoen's bi-directional phase adjustment

This approach to phase-adjustment 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** (3.1) 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:

$$\phi(t^+) = P(\phi(t)) = \phi(t) - \alpha \cdot sin(2\pi\phi(t)) \cdot |sin(2\pi\phi(t))| \tag{3.1}$$

, 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 [6].

The effects then of adjusting phases—upon hearing "fire"-events, according to this newest update-function (3.1)—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 (3.1), 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 $\phi(t) \approx 0.5 = \frac{1}{2}$.

3.2 Phase & frequency synchronization

GJØR: [Introduser det andre, og mer utfordrende, phase & frequency (ϕ-&ω) synchronization problemet, uthevet og i fet skrift. Deretter løsningene dens (Freq.-Adj.-metodene), som man må bruke i tillegg til ϕ-løsningene eller Phase-Adjustment-metodene. "This is the second and harder problem of synchronizing both phases ϕ_i, as well as frequencies ω_i, for all agents i more or less simultaneously."].

3.2.1 K. Nymoen's 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 $\omega_i(t^+)$:

$$\omega_i(t^+) = \omega_i(t) \cdot 2^{F(n)}, \tag{3.2}$$

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:

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 (3.3), plotted in Figure 3.1, 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.

$$\epsilon(\phi_i(t)) = sin^2(\pi\phi_i(t)) \tag{3.3}$$

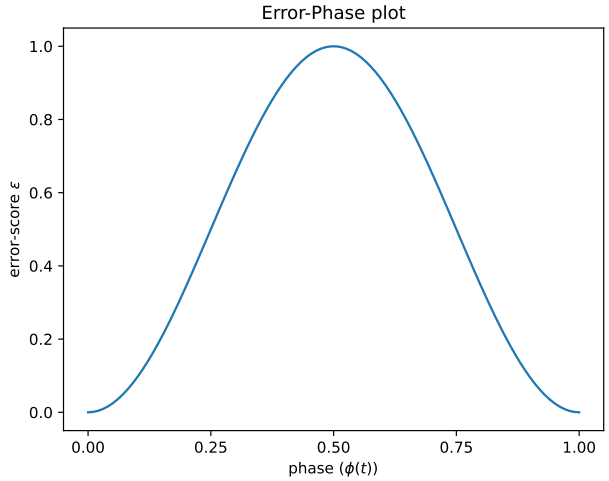


Figure 3.1: Error measurement (3.3) plotted as a function of phase

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 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 3.2, 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* $s(n)$.

Step 2: The first self-awareness component, s(n) This aforementioned self-assessed synchrony-score, $s(n)$, is in fact simply the median of error-scores ϵ.

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.

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.

$$\rho(\phi) = -sin(2\pi\phi(t)) \in [-1, 1] \tag{3.4}$$

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*.

Step 4: the H-value, and the H(n)-list The following value, being "frequency-update-contributions", is then (as previously mentioned) calculated immediately

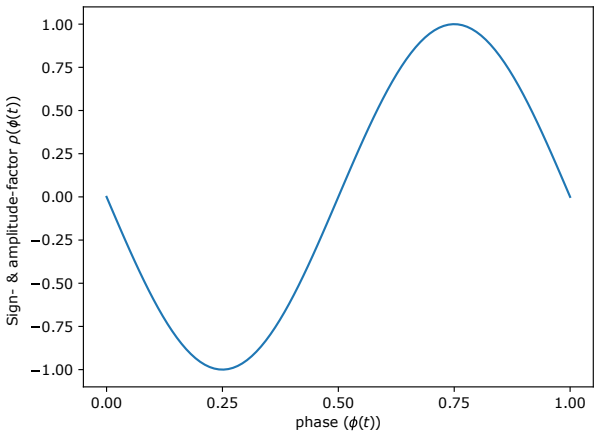


Figure 3.2: The amplitude- & sign-factor, $\rho(\phi(t))$, where $\phi(t)$ is the phase of an agent at time t when it heard "fire"-event n. Notice how the amplitude is equal to 0 when the phase is equal to 0,5.

when the agent perceives another agent's "flashing"-signal:

$$H(n) = \rho(n) \cdot s(n) \tag{3.5}$$

Here we then multiply the factor $\rho(n)$, depicted in Figure 3.2, 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 3.3. We hence see that the smallest value $H(n)$ 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 (3.3) and Figure 3.1) — 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).

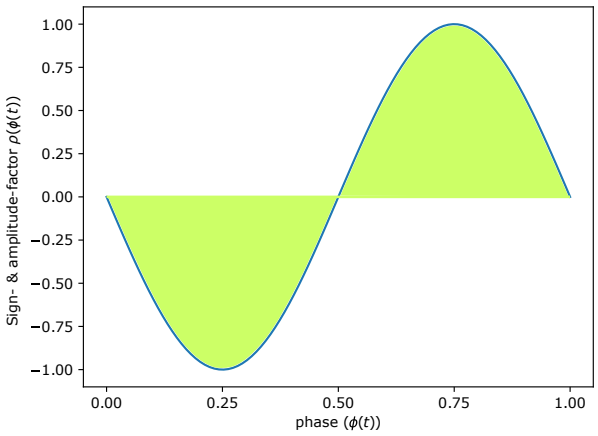


Figure 3.3: All possible $H(n)$-values marked in green, given by $\rho(n) \cdot s(n)$, where $\rho(n)$ is defined as above and depicted in Figure 3.2, and the self-assessed synch-scores $s(n) \in [0, 1]$. The $s(n)$-scores were again simply the median of m error-scores $\{\epsilon(n), \epsilon(n-1), \epsilon(n-2), \ldots, \epsilon(n-m)\}$ which again were given by the error-function (3.3) plotted in Figure 3.1. Observe, in conjunction with Equation (3.6), what the authors [6] point out when describing their frequency update function, when saying that they specify a function that decreases frequency if a fire event is received in the first half of a node's cycle (i.e. phase $\phi(t) < 0.5$), and speeds up if in the last half (to "catch up' with the firing node). Also note how the wanted design of the authors, of causing (close to) 0 change to the frequency half-way through the cycle, is enabled by how the "frequency-adjustment-contribution" $H(n)$ can be equal to 0 despite the reacting node having high error scores and a high $s(n)$-score.

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 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 error-scores.

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.

The final step: the frequency update function, $\omega_i(t^+)$ **BESKR.:** [Om F(n) og selve oppdateringen av frekvens].

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:

$$\omega_i(t^+) = \omega_i(t) \cdot 2^{F(n)}, \tag{3.6}$$

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

$$F(n) = \beta \sum_{x=0}^{k-1} \frac{H(n-x)}{k}, \tag{3.7}$$

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 (3.7), is a weighted average of all the agent's $H(n)$-values accumulated throughout the agent's last cycle.

3.3 System target state: harmonic synchrony

BESKR.: [Om target-staten til systemet, som oppfunnet av Kristian Nymoen et al. (?)].

The state of harmonic synchrony is defined [6] as the state in which all agents in the musical collective "fire"/"flash", as described in Subsection 5.1.2, 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 []. But somehow, 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 looks is shown visually in Section 5.4, especially in Figure 5.7.

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. [6] 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. 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, depicted in Figure 3.4b, where one can observe the presence of harmonics and overtones having frequencies with integer relationships to the fundamental (smallest) frequency at around 196 Hz.

More accurately then, and inspired by integer-relationships in the frequencies of harmonics and the fundamental frequency in the spectrogram of a waveform, K. Nymoen et al. [6] describe the *legal* frequencies the robots's oscillators in the musical robot collective have to adhere to in order for the oscillator-frequencies to be considered *harmonically synchronized*:

All musical robots i, in a harmonically synchronized state, will have frequencies ω_i which are element in the mathematical set

$$\Omega_{legal}(\omega_0) = \omega_0 \cdot 2^{\mathbb{N}_0} = \{\omega_0, 2\omega_0, 4\omega_0, 8\omega_0, ...\}, \tag{3.8}$$

where ω_0 is the lowest frequency in the robot-collective (or the fundamental frequency if you will), and $\mathbb{N}_0$ are the natural numbers including the number zero. If e.g. the smallest oscillator-frequency in the musical robot collective (ω_0) was equal to 1.5Hz, *legal* oscillator-frequencies the rest of the musical robots in the collective could have would be $\Omega_{legal}(1.5\text{Hz}) = \{1.5\text{Hz}, 3\text{Hz}, 6\text{Hz}, 12\text{Hz}, ...\}$.

Hence, in terms of oscillator-/robot-frequencies ω_i for all robots i, we have a harmonically synchronized robot-collective if (and only if)

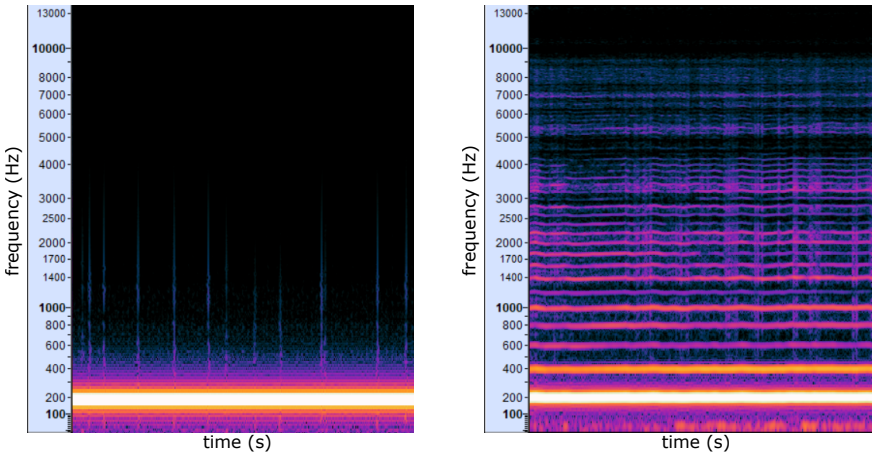
$$\omega_i \in \Omega_{legal}(\omega_0), \forall i, \tag{3.9}$$

where we then say all robots in the robot-collective have *legal* frequencies.

This state of *harmonic synchrony* is then the system goal state K. Nymoen et al. achieve using their phase- and frequency-update/-adjustment functions, as explained above in Section **??** and **??**, and is also the target/goal state we want to continue achieving in this thesis, but hopefully with even more *self-aware* methods (which are hopefully quicker and more robust).

3.3.1 Detecting harmonic synchrony

In order to test and evaluate synchronization performance in their firefly-inspired oscillator-system, K. Nymoen et al. [6] develop a measurement used to detect



(a) The frequency spectrogram of the audible waveform being a monotone and purely generated G3-tone at 195.99 Hz [9].

(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 in 3.4a with my voice.

Figure 3.4: Frequency spectrograms of two different-sounding waveforms of the same G3-tone at 195.99 Hz. Note the absence and presence of harmonics and overtones in waveform 3.4a and 3.4b respectively, as well as the integer-relationships between the fundamental (lowest) frequency and the harmonics in 3.4b. Frequencies in a harmonically synchronized agent collective will for the first ϕ-problem resemble the frequencies in 3.4a, where all frequencies are equal and constant. Conversely, when frequencies can be heterogenous and unequal, as in the ϕ- & ω-problem, the frequencies in a harmonically synchronized agent collective will rather resemble the frequencies in 3.4b, where these higher frequencies with integer-relationships to the fundamental and lowest frequency can be present.

when the system has reached a state of synchrony. Using the firings of the fireflies, some well-defined conditions have to be met in order for the fireflies to be deemed *harmonically synchronized*:

- **Condition 1**: Firing may only happen within a short time-period t_f.
- **Condition 2**: All nodes must have fired at least once during the evaluation period.
- **Condition 3**: Between each t_f, a period t_q without fire events must be equally long k times in a row.

By utilizing transmitted firings/pulses from the robots in our robot-collective, these conditions can be enforced and checked throughout the synchronization-process, in order to detect if the oscillator-network becomes harmonically synchronized.

For getting a better idea of how these conditions being met looks like, see the **Harmonic synchrony detection plot** in Figure 5.7 where the oscilla-

tors/robots fulfill the abovementioned requirements right before ending the synchronization process.

These requirements, amongst other illustrations in Nymoen et al.'s paper [6], thus constitutes a blueprint for the design of a performance or synchrony measurement able to detect the achievement of harmonic synchrony in a decentralized network of "firing"—or pulse-coupled—oscillators. The time having passed from the start of the synchronization-process until the detection of harmonic synchrony will then be defined as the performance score, indicating how fast or slow the oscillators are at synchronizing.

The exact details of how such a performance or synchrony measurement is implemented for our musical multi-robot oscillator-network, in the synchronization-simulator, will be given in Section 5.4.

Chapter 4

Tools and software

BESKR.: ["which describes what you've used" — ish Kyrre].

BESKR.: [Kan flyttes til en egen seksjon hvis dette kapittelet ikke ville vært så stort (jf. 'ThesisChecklist' på Robin-wikien)].

BESKR.: [(Hentet fra Tønnes sin master, om Tools and engineering) En introduksjon til de forskjellige verktøyene og prosessene brukt iløpet av masteroppgaven. Fokuser på fysisk arbeid gjort, og ingeniør-delene av masteroppgaven, inkludert 3D-design av de fysiske robotene, valg av deler, simulering i systemer, og testingen, valideringen, og verifikasjonsmetoder brukt i oppgaven. Gjerne også en oversikts-tabell av verktøy og programvare brukt].

4.1 Software

- Notepad++ v8.1.9.2 (64 bit). For writing my master's thesis and code.
- C#.
- Unity Version 2021.2.0f1. Unity is originally a game-development platform, but can also be used to make **INKL.:** [realistic]? simulations containing physical rigid-bodies using the <bla.bla Rigidbody>-physics engine.
- Python 3.10.0. for plotting and analysing data.
- Inkscape 1.1.1 for editing and creating vector graphics.
- Gimp 2.10.30 for editing and creating raster graphics.
- Audacity 3.1.0 for plotting frequency-spectra of recorded waveforms in Figure 3.4.
- LMMS 1.2.2, being a music-making system and digital synthesizer, for synthesizing harp-tones used for designing "fire"-signals.

4.2 Hardware

- CXT USB-Microphone for recording the hummed waveform, as in Figure 3.4b.

Chapter 5

Implementation

BESKR.: [What I've done / developed].

BESKR.: [Her presenterer jeg re-implementasjonen / etterlikningen / implementasjonsspesifikke valg av K. Nymoens approach og teori i det nye systemet / simulatoren min i Unity — samt hvordan jeg har verifisert at mekanismene fungerer (e.g. fase-synkroniseringsplott)].

This chapter gives an overview of the developed musical multi-robot system, methods implemented for it, as well as the performance measurement used to evaluate these methods with. 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 3.3.

These goals firstly require of the agents the modelling of oscillators with their properties, like phase and frequency, as explained further in Subsection 5.1.1. 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 5.1.2.

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. How the system target state of harmonic synchrony is detected will then be described in Section 5.4.

5.1 Simulator setup: the musical multi-robot collective

BESKR.: [Introduserer og presenterer det utviklede (simulator-)systemet og dets system-komponenter du har utviklet i Unity selv, veldig gjerne med et fint Unity-/Simulator-system-skjema].

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" [6]. 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 to each other and adjusting themselves accordingly, 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 5.1 and Figure 5.2.

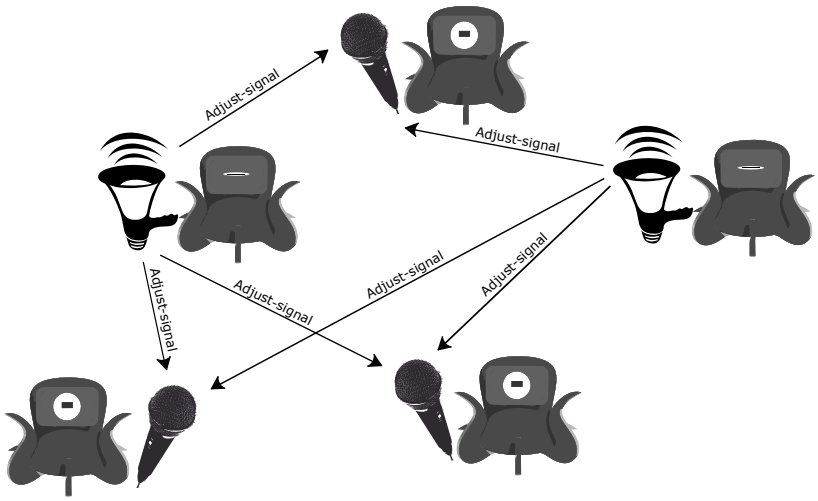
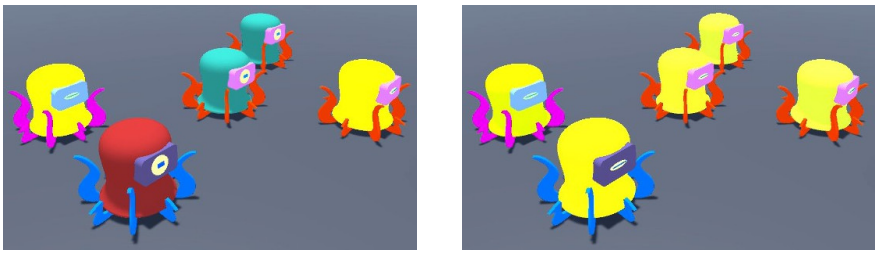


Figure 5.1: Illustration/Schematic: The musical robot collective entraining to synchronize to each other, or more specifically to achieve harmonic synchronization, through performing phase- & frequency-adjustments. Agents that are not firing at the moment will adjust themselves after hearing a transmitted "fire"-/adjustment-signal from a neighbouring firing agent.

Now, the synchrony simulator system components will be expounded and explained in detail.

5.1.1 The individual agent: a musical robot

BESKR.: [Om den enkelte roboten / agenten min med alle egenskaper den har osv.].



(a) Screenshot from the very beginning of a Synchronization-simulation.

(b) Screenshot from the same Synchronization-simulation as in 5.2a, a few moments later.

Figure 5.2: From simulation: An example of the system target goal of harmonic synchrony being achieved in a musical robot collective, where oscillator-frequencies ω are constant and equal (1Hz), or in other words synchronized already, but oscillator-phases ϕ are unsynchronized and initially uniformly random numbers in the range of $[0, 1]$.
At first in 5.2a, we see the agents firing, i.e. blinking with their eyes and turning their body yellow, asynchronously at first. Only two robots, one with pink and one with red tentacles, fire synchronously so far. Seconds later in 5.2b, after having listened to each others's adjustment-signals and adjusted themselves accordingly, all agents now fire simultaneously and synchronously.

As introduced and presented earlier, our musical robot collective will then consist of models of M. J. Krzyzaniak and RITMO's *Dr. Squiggles*, 3D-models of which can be seen in Figure 5.2.

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 5.1.

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.

Unless otherwise is stated, the heterogenous visual looks of the Dr. Squiggles robots in Unity have no real difference in the simulator and is only that (for visual looks), not implying other values or methods used.

5.1.2 Robot communication: the "fire"-signal

BESKR.: [Om kommunikasjonen til robotene mine: *audio / "fire" signalet*].

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*, as a way (discovered by K. Nymoen et al. [6]) to attain the system target goal of *harmonic synchrony*, to be elaborated upon in Section 3.3.

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*.

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 5.2 and Section 5.3.

5.1.2.1 Under the hood in the simulator

GJØR: [Fyll inn her med megafon-/mikrofon-pollingen hvis du implementerer det].

Pretty basic and kinda like poking another robot immediately after a phase-climax.

5.1.2.2 Audible to human simulator-observer

Fire signal design The fire-signal audible to the human observer watching and listening to the musical robots synchronizing to each other should be distinct and short enough so that one can clearly distinguish between which robots are firing and when. At the same time, the firing-sounds should to the human observer not be too sharp and loud to listen to so that it is uncomfortable observing the musical robot collective. Keeping these aspects in mind, some relatively soft and pleasant firing-sounds from an instrument usually associated with harmonious music—the harp—were produced and developed for the synchronization-simulator.

Some manually and empirically perceived harmonious (in terms of sounding good when played together) musical tones were digitally reproduced and recorded in the form of harp-plucks. This was achieved using the music-making system and digital synthesizer LMMS, and a digital string-instrument in the form of a LMMS-plugin from DSK Music (fotnote m/ link https://www.dskmusic.com/dsk-world-stringz-updated/). The musical tones reproduced with the digital harp-instrument were some of the tones perceived in the intro of Pogo's song "Stranger-

ous." (fotnote) One possible avenue to explore in order to find harmonious chords or tones—when played together—in a more automatic approach, live and online during simulation, is discussed in Further work in Chapter 7.

However, the harp-sounds produced with DSK's LMMS-plugin are primarily long and constant harp-plucks, as shown in Figure 5.3a. Using such harp-plucks as a fire-sound directly can make it hard to distinguish to the human ear when multiple robots are playing these frequently and simultaneously. The harp-sounds thus need to be slightly edited—which they in our implementation were in the audio-editing program Audacity—so that the long and constant harp-pluck became more of a "quick" and distinguishable "sound-bullet" — essentially what we want in a solid "fire"-signal sound. The "before" and "after" of such an audio-editing process is depicted in Figure 5.3. Edits performed on waveform 5.3a to obtain waveform 5.3b consists of effects like "Fade Out" (to dampen the sound in the tail of the waveform **INKL.:** [and get a shorter "sustain"]**?**, and hence obtaining a "quicker" sound), in the complete beginning "Amplify" (in order to get a as high "attack"/max-amplitude as wanted to make the fire-sound audible enough before it quickly decays) — and obviously cutting the waveform accordingly (from where it had amplitude ≈ 0 to the end of the waveform).

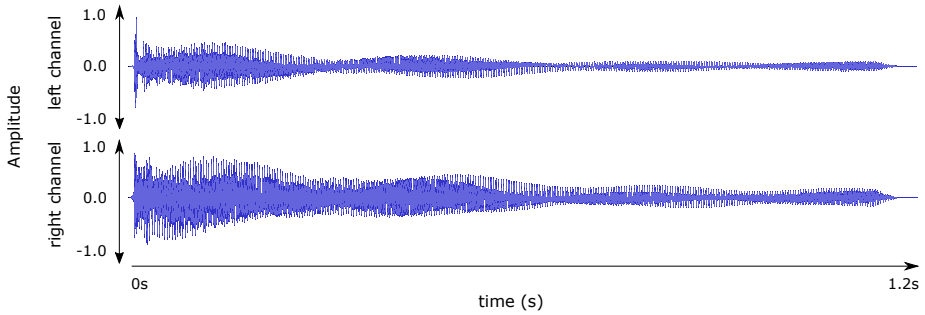
Frequency based fire-sound assignment **BESKR.:** [Presenting the assignment of firing-sounds, as developed in **Fire signal design**, (with varying fundamental frequencies) based on robot-frequencies].

GJØR: [Lag kjapt en skisse som i Logs&History-rM-notatet og kombiner dee med implementasjons-spesifikke detaljer som hvordan jeg fikk til å bytte frekvens].

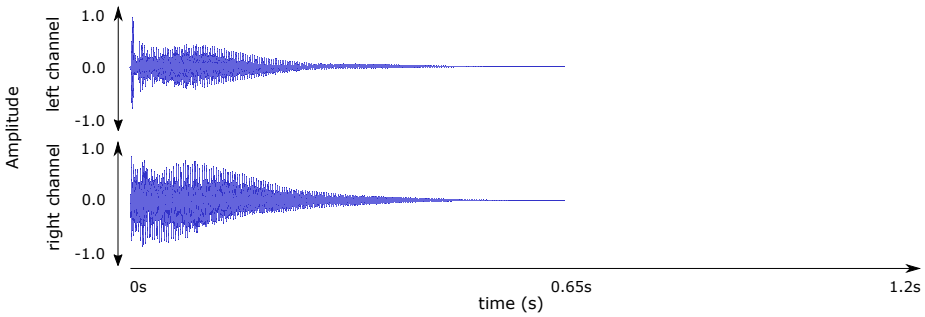
Given that our developed system is not simply a synchronization-system, which could have been implemented with simpler electrical circuits e.g. [], but also a musical robot system — the displayal and expression of the robots synchronizing and becoming synchronized should have a musical dimension, as well as providing a clear way to see whether the robots are getting synchronized or not. Therefore, a musically interesting and meaningful way of signalizing the synchronization-process was needed. The answers to the question of what is different between robots throughout the simulation plays a key part here; namely, e.g. the robot's oscillator-frequencies. Different musical tones were recorded (again with a harp-instrument) and later edited into "fire"-signals, as described above, with various signal lengths given their pitch. The idea was to assign, online and dynamically throughout the simulation-run, "fire"-signal-transformed musical tones with higher pitch (or waveform-frequencies) to musical robots with the highest oscillator-frequencies in the robot-collective.

5.2 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



(a) A harp-sound (stereo audio waveform).



(b) A fire-sound transformed/designed harp-sound.

Figure 5.3: Here we see the starting point and end result (before and after) of a fire-sound design process, where the longer harp-sound in 5.3a got edited and transformed into the more rapid—and amongst many other sounds like it played at the same time, more distinguishable—fire-sound in 5.3b.

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.

5.2.1 Verifying Mirollo-Strogatz's phase-adjustment

Mirollo-Strogatz's approach for synchronizing phases in oscillators, as introduced in 2.4.2.1, is implemented in the Unity simulator, and each agent is endowed with **phase update function** (2.3) with which they adjust themselves according to when perceiving a "fire"-signal as described above.

The verification that this works in the newly built synchronization-simulator was performed by dumping all agents's phase-values $\phi(t)$ during simulation-runs. A plot of these $\phi(t)$-values, evolving through simulation-time in seconds, is shown in Figure 5.4.

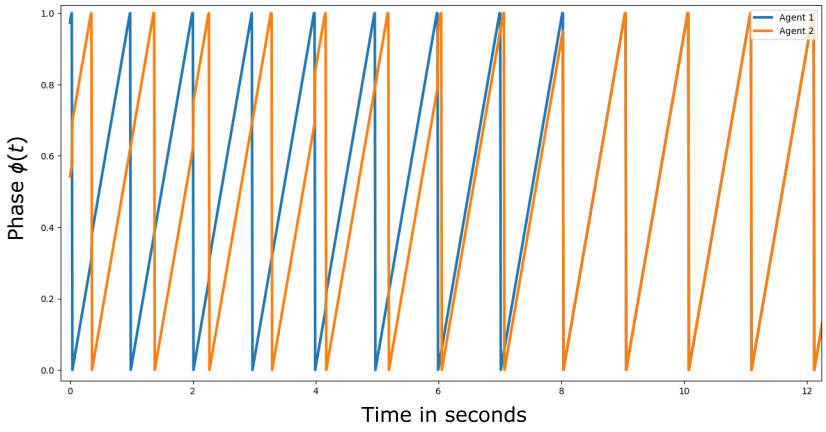


Figure 5.4: "Standard" phase-adjustment with Mirollo-Strogatz's approach

5.2.2 Verifying K. Nymoen's bi-directional phase-adjustment

K. Nymoen et al.'s approach for synchronizing phases in oscillators, as introduced in Section **??**, is implemented in Unity, and each agent is endowed with **phase update function** (3.1) with which they adjust themselves according to when perceiving a "fire"-event as described above.

The verification that this works in the newly set-up simulator-environment was performed by analysing carefully all the agents's phase-values $\phi(t)$ throughout a simulation-run. Such an analysis/plot can be seen in Figure 5.5.

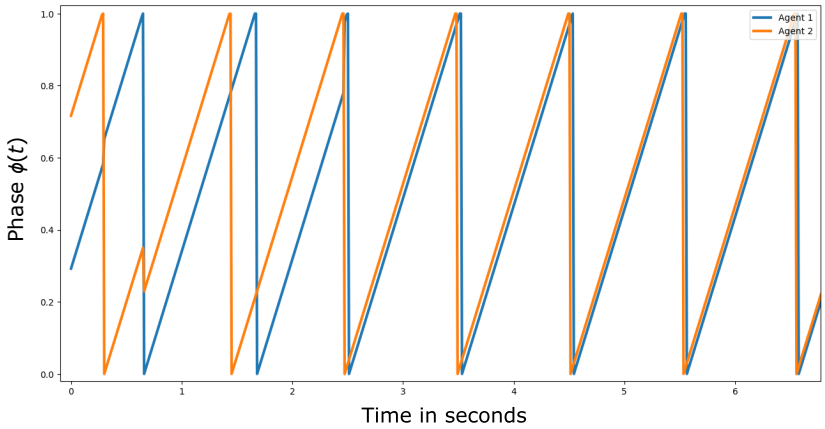


Figure 5.5: Bi-directional phase-adjustment with K. Nymoen et al.'s approach

5.2.3 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) [4]. 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).

].

5.3 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 3.3, 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 5.2, 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 3.3.

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

5.3.1 Verifying K. Nymoen's frequency-adjustment

In the newly proposed Unity simulator environment, the previously introduced self-assessed synch-score $s(n)$ (in 3.2.1) is implemented as 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):

$$errorBuffer = \{\epsilon(n), \epsilon(n-1), ..., \epsilon(n-m)\}, \tag{5.1}$$

then leading to:

$$\begin{aligned} s(n) &= median(errorBuffer) \\ &= median(\{\epsilon(n), \epsilon(n-1), ..., \epsilon(n-m)\}) \in [0,1], \end{aligned} \tag{5.2}$$

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.

Regarding the "frequency-update-contributions" (the H-values described in 3.2.1) in my Unity-simulator, all the calculated H-values are 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.

By using K. Nymoen et al.'s frequency adjustment method, synchronization of initially random frequencies in our newly developed Unity simulator is achieved, like can be seen recorded in Figure 5.6.

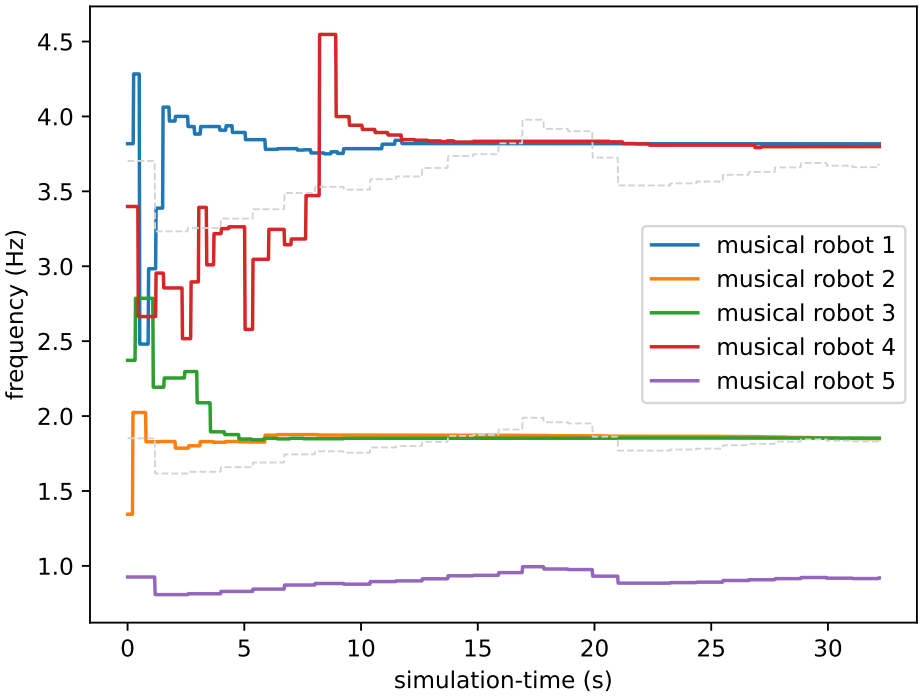


Figure 5.6: Frequency synchronization for five robots in a Unity synchronization simulation run achieving harmonic synchrony after 30 seconds. Note how the legal harmonic frequencies (dashed gray lines) are defined by the lowest—fundamental—frequency ω_0 in the robot collective fluctuating slightly below 1Hz, correctly leading to the legal frequencies right below 2 and 4 Hz.

5.3.2 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) [4]. 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).

].

5.4 Detecting harmonic synchrony

The performance measurement will be used in our synchrony-simulator to evaluate and test the multi-robot collective's ability to harmonically synchronize to each other. As mentioned in Subsection 3.3.1, K. Nymoen et al.'s requirements and illustrations [6] for achieving *harmonic synchrony* serve as a blueprint or guide for how to similarly implement our synchrony or performance measurement. This performance measurement should be able, during synchronization-simulation, to detect if harmonic synchronization has been achieved in our decentralized oscillator-network. The successful triggering of this detection will then in turn terminate the synchronization simulation-run and save to a dataset the time it took to synchronize (the performance score), in the case of a 'synchronization-success' — an example of which can be seen in Figure 5.7.

The resulting and corresponding performance scores obtained using this performance measurement will then take values of the simulation time (s) it takes for the robot collective, from the start of the synchronization simulation, to achieve the system target state of *harmonic synchrony*, as specified in Section 3.3.

Now if **Conditions 1-3** from Subsection 3.3.1 are kept, we have harmonic synchrony.

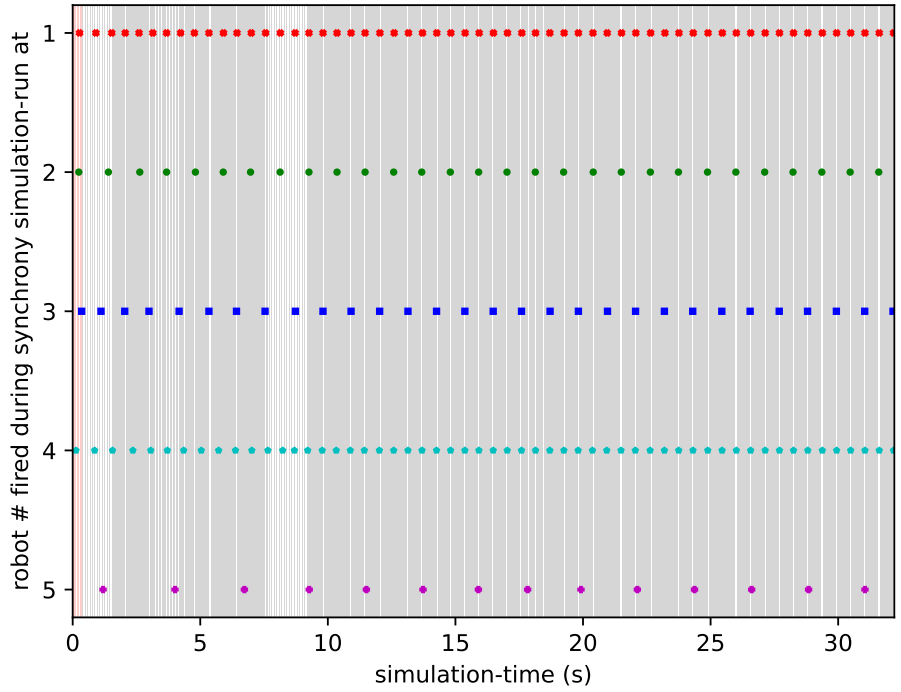


Figure 5.7: **Harmonic synchrony detection plot**: recording a detection of harmonic synchrony. The steps and events of **I)** through **VII)** can also be seen commented in pseudocode []. Initial red area represents the "start-up"-period where no (gray) t_q-value is defined yet. **I)** First robot firing, initiating a process of defining such a "silent" time-window t_q within which no nodes are allowed to fire, and a whole "legal-firing" t_f-window (here shaded in white and only 80ms long, as in [6]) ensues. **II)** At least one robot firing, and with their firing-time(s) giving us the *early* t_q*-defining* time-value (see schema below), as well as triggering second whole t_f-window to ensue. **III)** At least one robot firing after the *early* t_q*-defining* time-value was found, hence (with their firing-time(s)) giving us the *late* t_q*-defining* time-value right before using these when finishing the on-started process of defining the "silent" time-window t_q; triggering a half (due to stabilization purposes by future centering of fire-events) t_f-window to ensue. **IV)** A robot is caught firing illegally during a "silent" t_q-window and hence resets the *towards_k_counter*-variable, as well as (re-) starts a process like in **I)** for (re-) defining a new t_q-value. **V)** Firing after this point only happens during short "legal-firing" t_f-windows, so **Condition 1** (cf. 3.3.1) is held. **VI)** All robots have fired at least once during the evaluation-period, so **Condition 2** is held. **VII)** The robot collective has fired legally and evenly, without resetting t_q, k (equal to 8 in this case) times in a row, so **Condition 3** is fulfilled — and harmonic synchrony is thus achieved and detected after 21.9 seconds of synchronization.

My specific implementation of the synchrony measurement essentially consists of enforcing all the requirements or rules listed in 3.3.1, given some constant t_f- and k-values (e.g. $80ms$ and 8 respectively [6]). And again—to recall from 3.3.1—t_f is the short time-window within which nodes are allowed to fire at each beat, and k represents how many times nodes have to fire at even underlying pulses/beats in a row without changing the t_q-period—before becoming harmonically synchronized.

The requirement of firing evenly k times in a row with identical t_q-periods can be—and in fact is in our implementation—enforced by incrementing an integer variable *towards_k_counter* after a 'legal' t_f-window has occured (i.e. one or more nodes fired inbetween the onset and ending of the t_f-window), and conversely by resetting *towards_k_counter* to 0 when an illegally transmitted firing was heard during a 'silent' (or so it was supposed to be at least) t_q-window, hence restarting the synchrony-detection process—as can be seen occuring several times in Figure 5.8. Note that in this specific simulation run above, the agents were on their way to achieve harmonic synchrony five times before the 10th second of the synchronization-simulation already, but since one or more of them fired 'illegally' (i.e. inside a t_q-window), they were consequently 'punished'—or rather deemed 'not synchronized enough yet'—by getting their counter reset to 0. Eventually however, through further phase- & frequency-synchronization, the multi robot collective was in this case after 12.5 seconds able to achieve harmonic synchrony, when *towards_k_counter* became equal to k, as well as all other requirements for achieving *harmonic synchrony* was met. Note that this gives us a sense of how synchronized the robot collective is over time; the more even beats the robots have in a row, the more synchronized they are.

Initially, the t_q-period/-window is not initialized, as it entirely depends on the frequencies to which the robot-collective converges to; however, when an illegal firing (i.e. a firing perceived during a t_q-window) occurs—t_q is also then reset itself to a hopefully more correct value, given by the following formula, which is visually explained further in the schema in Figure **??**:

$$\begin{aligned} t_q^* &= late\_t_q^*\_defining\_timevalue - early\_t_q^*\_defining\_timevalue - t_f \\ &= median(\grave{t_1}, \grave{t_2}, ..., \grave{t_m}) - median(t_1, t_2, ..., t_n) - t_f, \end{aligned} \tag{5.3}$$

where n fire-events were recorded during the earlier t_f-window, and m fire-events were recorded during the later t_f-window.

The specific steps and procedures needed to implement this harmonic synchrony detection is expounded in detail in Algorithms 1, 2, and 3.

Algorithm 1: Harmonic synchrony detection part A (må evt. fylles inn)

Algorithm 2: Harmonic synchrony detection part B (må evt. fylles inn)

Algorithm 3: Harmonic synchrony detection part C (må evt. fylles inn)

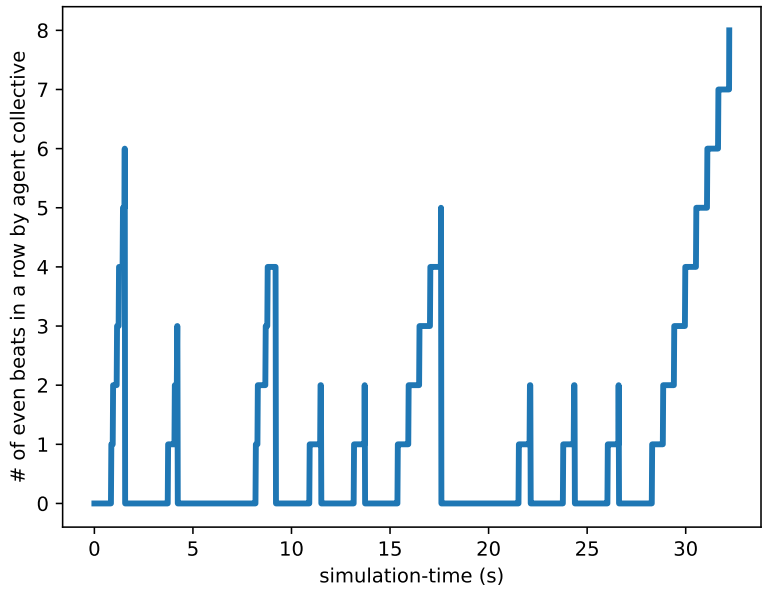


Figure 5.8: A **synchrony evolution plot**, displaying the temporal recording of the *towards_k_counter* variable throughout a synchrony simulation-run in Unity. The counter is incremented as the robot collective fires evenly within 'legal' t_f windows, and is conversely reset to 0 if illegal firings during 'silent' t_q windows are heard.

If a certain amount of time, e.g. 5 minutes [6], has gone without the detection of harmonic synchrony occuring, the simulation-run is terminated as a "fail". Hey hey there.

Chapter 6

Experiments and results

BESKR.: [Et kapittel der du har satt opp (med hyper-parametere og miljøvariabler f.eks. i en oversiktlig tabell) eksperimenter, kjørt simulation-runs med disse verdiene, og viser hva resultatene ble. Resultater kan være **performance plot**s (ift. harmonic synch.-times for diverse hyper-parametere og miljøvariabler (samt om robotene er homogene eller heterogene og isåfall hvordan), der plottet må lages spesifikt for hvert eksperiment, og kan f.eks. være boxplot eller tabeller med performance-/hsynchtime-/simulation-time(s)-verdier), **harmonic synchrony detection plot**s (altså hvordan harmonic synch. ble detectet i et simulation-run), **phase-&frequency-plot**s (altså hvilke fase- og frekvensverdier robotene hadde iløpet av simulation-run'et, via *plot_PhaseFrequencyPlot_for_SimRun.py*), eller **synchrony-evolution plot**s (der 'towards_k_counter'en iløpet av simulation-run'et plottes, via *plot_SynchronyEvolutionPlot_for_SimRun.py*)].

BESKR.: [Helst gode eksperimenter som motiveres og forklares, settes opp, og utføres m/resultater man diskuterer og analyserer. Det er bra å evaluere fra flere synspunkt med flere research methods og hvis tid].

This chapter presents the experiments and results set up and performed in the novel synchronization simulator in Unity, as presented in Chapter 5, for certain configurations of musical robot collectives. Effects of the individual musical robots's hyperparameters on the collective achievement and performance of achieving harmonic synchrony are presented. Some examples are hyperparameters which determines how much each musical robot will adjust itself after hearing a transmitted fire signal from a neighbouring robot; α for phase adjustment and β for frequency adjustment.

The main performance scores presented in this chapter will consist of synchronization times / simulation time (s) (i.e. how long it takes robots to reach the state of harmonic synchrony if they ever do), accompanied by the respective and corresponding error rates during the belonging simulation runs (i.e. the percentage of robot collectives out of e.g. 30 runs failing to reach harmonic synchrony before the maximum time limit of e.g. 5 minutes).

6.1 Solving the ϕ synchronization problem

This is the section where experiments attempting to solve the first and simpler problem, namely synchronizing only the phases ϕ_i of all agents i, are presented and analyzed. These are then experiments where all musical robots have an equal and fixed frequency, only adjusting phases, in order to entrain to synchronize their phases to each other until reaching harmonic synchrony.

6.1.1 Reproducing K. Nymoen's phase synchronization

In order to see that our developed synchronization simulator in Unity yields more or less the same results as K. Nymoen et al.'s firefly system [6], similar experiments as reported in their paper are performed. These tell us whether differences in performance, in terms of synchronization times (s), is simply due to implementation differences, or actually because of the utilized synchronization methods and hyperparameters in question.

First off, and as performed and reported in K. Nymoen et al.'s paper [6] (c.f. their Fig. 7), Mirollo & Strogatz's phase adjustment method (as presented in 2.4.2.1) is experimented with for varying phase coupling constants α, as seen in Figure 6.1.

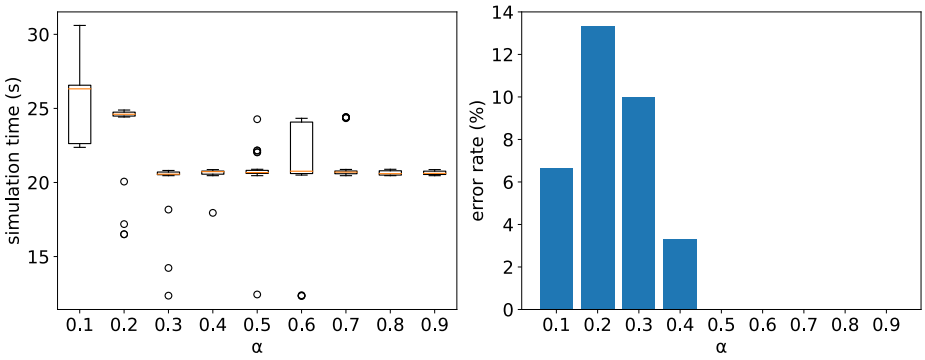


Figure 6.1: Synchronization times (s) for 6 robots with initially random and unsynchronized phases but equal and fixed frequencies (1Hz), for varying phase coupling constants α. 30 simulation runs per α. $\beta = 0$, $t_{ref} = 50ms$, $t_f = 80ms$, and $k = 8$. Maximum time limit was 5 minutes.

6.1.2 Hyperparameter tuning

Here we tune the hyperparameters α and $t_{ref\_percentage\_of\_period}$ for several robot collective sizes, according to performance scores of how long it takes robot collectives on average to achieve harmonic synchrony. Exactly these two specific hyperparameters are experimented with since they empirically seem to be the most important ones to set correctly before starting the simulator; that is, in order for the robots to actually manage achieving harmonic synchrony.

The specific choices of hyperparameter values to test synchronization times for, when wanting to tune the phase coupling constant α and the dynamic refractory period variable $t_{ref\_percentage\_of\_period}$, were chosen based on empirical

collective size = 3	$\alpha = 0.001$	$\alpha = 0.01$	$\alpha = 0.1$	$\alpha = 0.8$
$t_{ref\_percentage\_of\_period} = 0.03$	x	x	x	
$t_{ref\_percentage\_of\_period} = 0.05$	x	x	x	
$t_{ref\_percentage\_of\_period} = 0.1$	x	x	x	
$t_{ref\_percentage\_of\_period} = 0.5$	x	x	x	
collective size = 10	$\alpha = 0.001$	$\alpha = 0.01$	$\alpha = 0.1$	$\alpha = 0.8$
$t_{ref\_percentage\_of\_period} = 0.03$	x	x	x	
$t_{ref\_percentage\_of\_period} = 0.05$	x	x	x	
$t_{ref\_percentage\_of\_period} = 0.1$	x	x	x	
$t_{ref\_percentage\_of\_period} = 0.5$	x	x	x	
collective size = 25	$\alpha = 0.001$	$\alpha = 0.01$	$\alpha = 0.1$	$\alpha = 0.8$
$t_{ref\_percentage\_of\_period} = 0.03$	x	x	x	
$t_{ref\_percentage\_of\_period} = 0.05$	x	x	x	
$t_{ref\_percentage\_of\_period} = 0.1$	x	x	x	
$t_{ref\_percentage\_of\_period} = 0.5$	x	x	x	
collective size = 50	$\alpha = 0.001$	$\alpha = 0.01$	$\alpha = 0.1$	$\alpha = 0.8$
$t_{ref\_percentage\_of\_period} = 0.03$	x	x	x	
$t_{ref\_percentage\_of\_period} = 0.05$	x	x	x	
$t_{ref\_percentage\_of\_period} = 0.1$	x	x	x	
$t_{ref\_percentage\_of\_period} = 0.5$	x	x	x	

Experiment 6.1.2 results: Tuning hyperparameters α and $t_{ref\_percentage\_of\_period}$ for various musical robot collective sizes in the simpler phase synchronization problem. 30 simulation runs per α and $t_{ref\_percentage\_of\_period}$ pair. $t_f = 80ms$, and $k = 8$. Maximum time limit was 5 minutes.

hunches (gotten during observation of the synchronization simulator in Unity when playing with the hyperparameters) for what might might be acceptable, according to experience, in order to facilitate stable and successful synchronization simulation runs—for varying robot collective sizes.

An hypothesis and aforementioned hunch is e.g. that larger robot collectives do not require as large of a phase coupling constant α in order to synchronize to each other compared to that which smaller robot collectives require; as we remember α tells each robot how much to adjust its oscillator phase when hearing a "fire" signal, and when more neighbours are present to transmit "fire" signals to a robot—a larger quantity of small adjustments can in theory be equivalent to a few large adjustments. This is something we want to investigate more quantitatively and thoroughly by the following experiment.

In the Experiment 6.1.2 results, as seen below, various musical robot collectives's performance in terms of mean simulation times (s) from start of synchronization until harmonic synchrony is detected, along with their standard deviations, are presented. Again, since we are solving the phase (ϕ) synchronization problem, now only phases are initially unsynchronized, and frequencies are fixed and constant (1Hz) throughout the simulation runs.

6.1.3 Comparing phase adjustment methods

Now, an experiment follows where we investigate the validity of the claimed [] benefits of performing bi directional phase adjustments—both inhibitory and excitatory—compared to simply adjusting phases in an excitatory way (i.e. only "pushing" other ocillators's phase further or higher when firing, never "holding" or "dragging" it back).

6.2 Solving the $\phi\&\omega$ synchronization problem

This is the section for the experiments attempting to solve the second and harder problem of synchronizing both phases ϕ_i, as well as frequencies ω_i, for all agents i. These are then experiments where all musical robots originally have unequal and ever-changing phases and frequencies, adjusting both phases and frequencies in order to entrain to synchronize their phases and frequencies until reaching harmonic synchrony.

6.2.1 Reproducing K. Nymoen's phase and frequency synchronization

Here we present attempts made in the novel Unity synchronization simulator at recreating K. Nymoen et al.'s first results with their novel frequency synchronization method, which is utilizing, amongst other aspects, self awareness [6].

Ordering by phase couplings Given that K. Nymoen et al. do not mention their β value in their frequency synchronization experiment where they order for different phase coupling values α, an *empirically decent* β value of 0.4 is chosen for this experiment. What *empirically decent* refers to in this case are K. Nymoen et al.'s findings in the results of their last experiment [6] where synchronization times for various β values were evaluated; deeming $\beta = 0.4$ to be a good value, with no further improvement in synchronization performance when β is increased further. Also, when empirically observing the visual and aural signals being transmitted through the synchrony simulator in Unity, this value of $\beta = 0.4$ seemed to lead to relatively stable—given original instability—simulation runs where the robot collective does manage to achieve harmonic synchrony, which is desirable when testing how fast they can synchronize.

Here, K. Nymoen et al.'s self aware frequency adjustment, as implemented in the novel synchrony simulator in Unity, is experimented with for varying phase coupling constants α as in Figure 6.1 — only that this time not only phases are initially unsynchronized; robot frequencies are also initially unsynchronized. See the results in Figure 6.2.

Ordering by frequency couplings Now, we perform the same phase & frequency synchronization experiment as in Figure 6.2, except that this time we will fix the phase coupling constant α and instead test how the individual musical robots's various frequency coupling constants β affect the performance of the musical robot collective. These results are shown in Figure 6.3

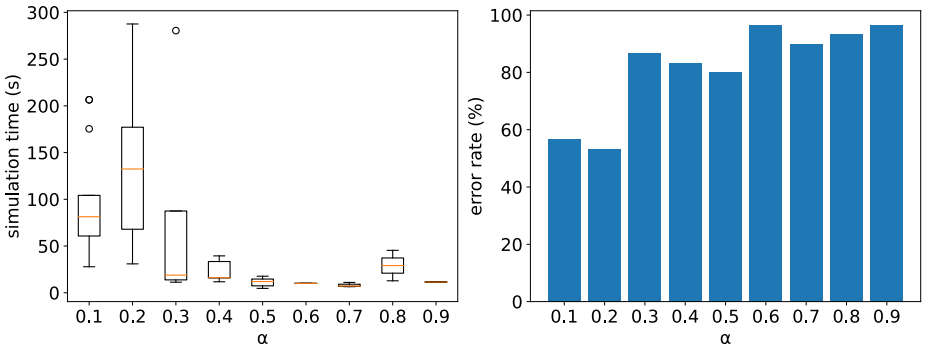


Figure 6.2: Synchronization times (s) for 6 robots with both initially random and unsynchronized phases, and frequencies, for varying phase coupling constants α. 30 simulation runs per α. $t_{ref} = 50ms$, $m = 5$, $t_f = 80ms$, and $k = 8$. Maximum time limit was 5 minutes.

Again, K. Nymoen et al. does not specify exactly the phase coupling constant they use in this latter experiment when testing their firefly-inspired synchronization system for various β values. Hence, the now fixed phase coupling constant α is here selected by reusing the α value found in the similar experiment presented in Figure 6.3 to yield the lowest error rate in the musical robot collective when synchronizing to each other. This then gives us a fixed $\alpha = 0.2$.

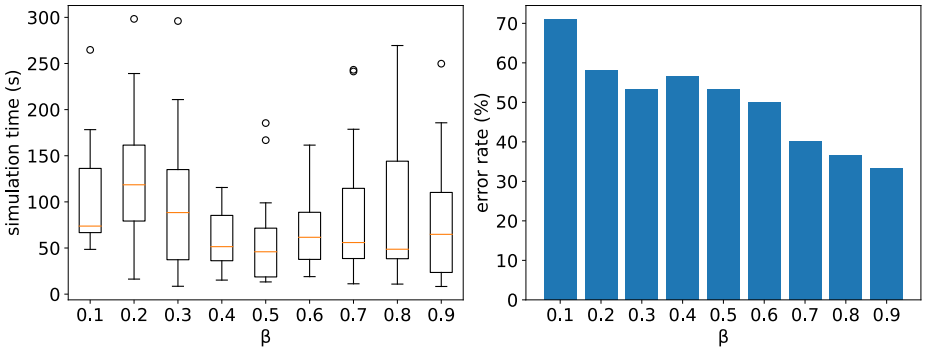


Figure 6.3: Synchronization times (s) for 6 robots with both initially random and unsynchronized phases, and frequencies, for varying frequency coupling constants β. 30 simulation runs per β. $t_{ref} = 50ms$, $m = 5$, $t_f = 80ms$, and $k = 8$. Maximum time limit was 5 minutes.

6.2.2 Hyperparameter tuning

Here we tune the hyperparameters β and m for several robot collective sizes according to performance scores. The reason for tuning exactly these two hyperparameters, the frequency coupling constant β and the error score list length (or memory length) m, is again since they seem the most significant for the synchronization performance.

6.2.3 Increasing degree of self awareness

Here the hypothesis of whether increasing musical robots's degrees of self awareness, specifically referring to the robots's *self awareness scope* [4], will affect the synchronization performance or not. This is tested in Unity for the more challenging $\phi\&\omega$ problem of harmonically synchronizing both phases and frequencies. Perhaps a larger self awareness scope, meaning more knowledge about the social environment, will lead to the robots having a better "overview" of the environment; hence leading to shorter simulation time (s) before reaching the goal state of harmonic synchrony. Or perhaps hearing more "fire" signals on average simply will be disturbing to the robots and hence disturb and slow down their entrainment towards harmonic synchrony. This experiments attempts to answer questions like these by for the following three scenarios evaluating collective synchronization performance:

1. **Minimal self awareness scope**: Each individual robot only hears the nearest neighbouring robot's "fire" signals []. In this case, robots have limited and more local knowledge.

2. **Radial self awareness scope**: Each individual robot hears neigbouring robots's "fire" signals within a radius d around it []. In this case, robots have more knowledge but also still locally.

3. **Global self awareness scope**: Each individual robot hears all other neigbouring robots's "fire" signals []. In this case, robots have maximum and global knowledge when it comes to awareness of other neighbouring robots.

In this way, the degree to which robots are self aware of or communicating with other robots is increasing. The effects of increasing the degree of self awareness in this sense are shown in [].

Chapter 7

Conclusions

BESKR.: [Where I shall follow-up on my *research questions* by a discussion of to which degree—and in what ways—the thesis-/project-work has answered them].

GJØR: [Se på (for kapittel-inspirasjon):

- Tønnes . MSc-thesis . Discussion-Ch.
- Jim . 'how to write a master thesis.pdf' . 'Conclusions'.

].

7.1 General discussion

7.2 Conclusion

7.3 Further work

Automatic and online (harmonic) fire sound generation. As alluded to in 5.1.2.2, one possible avenue to explore in order to find harmonious chords or tones—when played together—in a more automatic approach, live and online during simulation, could be found through musical machine learning models predicting fitting and accompanying chords and harmonies (given a tone) in real-time, like the ones tested thoroughly in B. Wallace's interactive music system PSCA [10]. This was however not attempted in this thesis, as this was beyond the scope of the project.

Physical domain. Going from simulation to the real physical world, using M. J. Krzyzaniak and RITMO's actual musical robots, the Dr. Squiggles (fotnote) [7]. Using an audio interface with the Dr. Squiggles robots in order to transmit or play audible "fire" signals would perhaps correspond to *self awareness scope 2* as defined in this thesis.

The main reason why these ideas were not possible to explore throughout this thesis work in the end, was due to an (to me at least) unseen and—for

months—unobservable Unity peculiarity (perhaps only to a Unity novice like me) which in turn caused visible bugs in two mechanisms—namely the harmonic synchrony detection mechanism as well as my implementation of K. Nymoen et al.'s frequency adjustment method. So given that I only saw problems in the harmonic synchrony detection and frequency synchronization, I ended up spending considerable time throughout the thesis period trying to debug these two mechanisms, without success—or if any then with more bugs popping up. The way out of this extremely confusing and frustrating situation reared its head eventually as my attention was brought to the synchrony simulator's determinism. Little did I know that once I made my synchrony simulator deterministic, the problems I had been debugging for months were solved immediately. And after discussing with supervisors, and a much more experienced Unity user (Frank Veenstra), the root (and to me unseen) problem, and the reasons why it lead to those visible problems in my harmonic synchrony detection and frequency synchronization, suddenly made total sense. And so now this resulting knowledge about indeterminism in Unity, learnt the hard way, will most likely be shared through a ROBIN wiki page or the likes. Hopefully, this will prevent future UiO robotics master's students from stumbling onto the same problems I did, and from ending up in similar rabbit holes as I ended up in.

Bibliography

[1] Daniel Goldman, Haldun Komsuoglu, and Daniel Koditschek. "March of the sandbots". In: *IEEE Spectrum* 46.4 (2009). Publisher: IEEE, pp. 30–35.

[2] Auke Jan Ijspeert. "Central pattern generators for locomotion control in animals and robots: a review". In: *Neural networks* 21.4 (2008). Publisher: Elsevier, pp. 642–653.

[3] nature journal. *Biomimetics*. URL: https://www.nature.com/subjects/biomimetics (visited on 03/30/2022).

[4] 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.

[5] 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.

[6] 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.

[7] 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).

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

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

[10] Benedikte Wallace. "Predictive songwriting with concatenative accompaniment". Master's Thesis. 2018.