Chapter 1

dummy-hack Introduction

Chapter 2

dummy-hack Background

Chapter 3

Baseline

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

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

3.3 System target state: harmonic synchrony

The state of harmonic synchrony is defined [2] 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 can look is shown in Section 4.4, especially in Figure 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. [2] reason when discussing their own interactive "Firefly" music-system, as well as coining the term of *harmonic synchrony*:

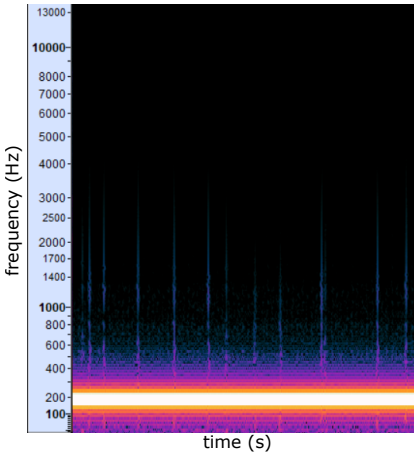
Temporal components in music tend to appear in an integer-ratio relation to each other (e.g., beats, measures, phrases, or quarter notes, 8ths, 16ths).

and

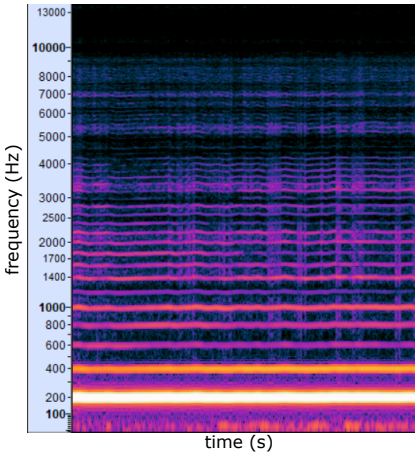
Being an interactive music system, people may want their device to synchronize with different subdivisions of a measure (e.g. some play quarter notes while others play 8ths).

Accomodating for these aspects then, K. Nymoen et al. took inspiration for achieving synchronization in a decentralized system from the concept of *har-*

monics 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 3.1b, where one can observe the presence of harmonics and overtones having frequencies with integer relationships to the fundamental (smallest) frequency at around 196 Hz.



(a) The frequency spectrogram of the audible waveform being a monotone and purely generated G3-tone at 195,99 Hz [3].



(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 3.1a with my voice.

Figure 3.1: 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 3.1a and 3.1b respectively. Frequencies in a harmonically synchronized agent collective will for the first ϕ-problem resemble the frequencies in 3.1a, 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 3.1b, where these higher frequencies with integer-relationships to the fundamental and lowest frequency can be present.

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. [2] 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

$$\Omega_{legal}(\omega_0) = \omega_0 \cdot 2^{\mathbb{N}_0} = \{\omega_0, 2\omega_0, 4\omega_0, 8\omega_0, ...\}, \tag{3.1}$$

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)

$$\omega_i \in \Omega_{legal}(\omega_0), \forall i, \tag{3.2}$$

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 3.1 and 3.2, 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. [2] develop a measure used to detect 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*:

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

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 *performance-measure plot* in Figure 4.5 where the oscillators/robots fulfill the abovementioned requirements right before ending the synchronization process.

These requirements, amongst other illustrations in Nymoen et al.'s paper [2], thus constitutes a blueprint for the design of a performance-/synchrony-measure 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-/synchrony-measure is implemented for our musical multi-robot oscillator-network, in the synchronization-simulator, will be given in Section 4.4.

Chapter 4

Implementation

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 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 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 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. How the system target state of harmonic synchrony is detected will then be described in Section 4.4.

4.1 Simulator setup: the musical multi-robot collective

BESKR.: [Introduserer og presenterer det utviklede (simulator-)systemet 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" [2]. 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 4.1 and Figure 4.2.

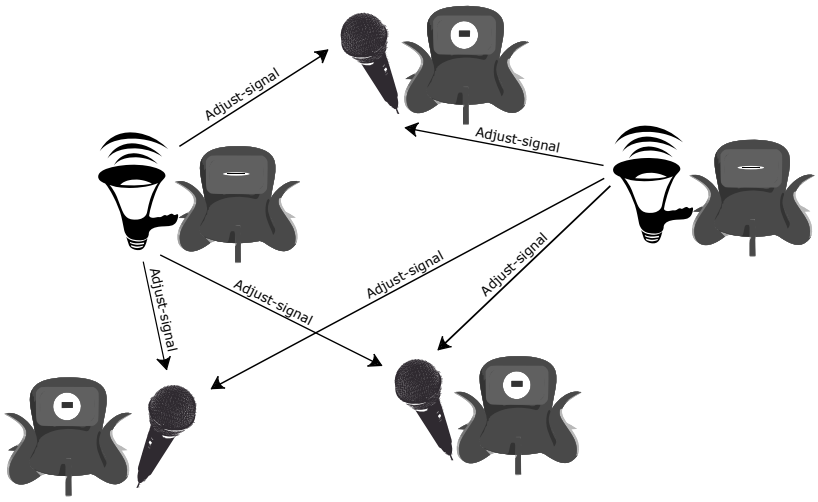
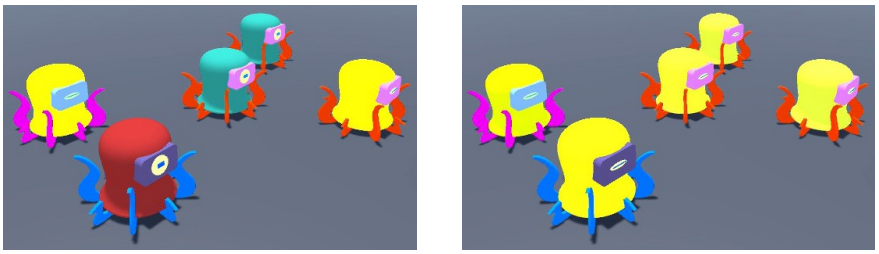


Figure 4.1: Illustration/Schematic: The musical robot collective entraining to synchronize to each other, or more specifically to achieve harmonic synchronization, through performing phase- & frequency-adjustments. Agents that are not firing at the moment will adjust themselves after hearing a transmitted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

4.1.1 The individual agent: a musical robot

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 4.2.

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



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

(b) Screenshot from the same Synchronization-simulation as in 4.2a, a few moments later.

Figure 4.2: 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 4.2a, 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 4.2b, after having listened to each others's adjustment-signals and adjusted themselves accordingly, all agents now fire simultaneously and synchronously.

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*, as a way (discovered by K. Nymoen et al. [2]) 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 4.2 and Section 4.3.

4.2 Synchronizing oscillator-phases

GJØR: [Introduser det første ϕ-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."].

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 Verifying Mirollo-Strogatz's phase-adjustment

Mirollo-Strogatz's approach for synchronizing phases in oscillators, as introduced in **??**, is implemented in the Unity simulator, and each agent is endowed with **phase update function** (**??**) 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 4.3.

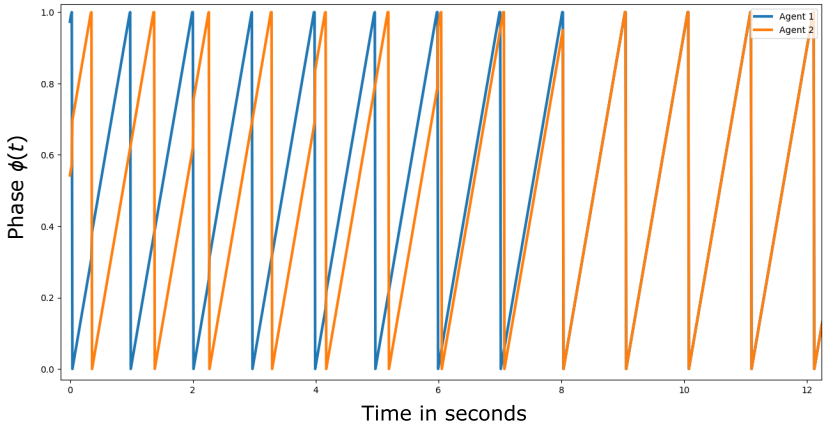


Figure 4.3: "Standard" phase-adjustment with Mirollo-Strogatz's approach

4.2.2 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** (**??**) 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 4.4.

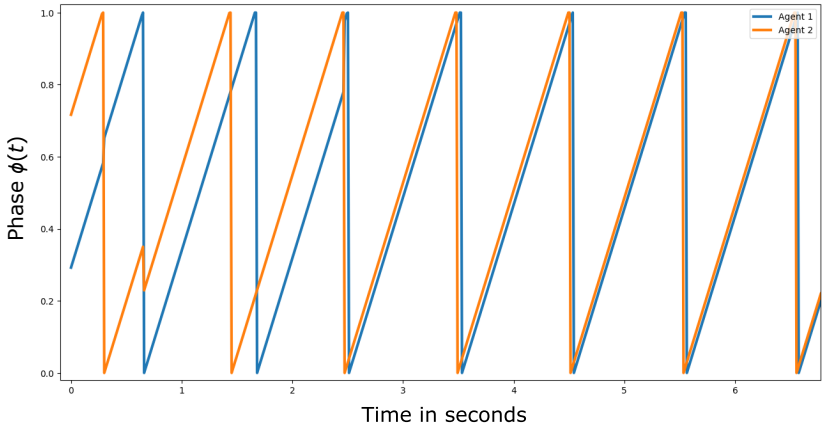


Figure 4.4: Bi-directional phase-adjustment with K. Nymoen et al.'s approach

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

GJØR: [Introduser det andre, og mer utfordrende, ϕ-&ω-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."].

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 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 3.3.

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

4.3.1 Verifying K. Nymoen's frequency-adjustment

In the newly proposed Unity simulator environment, the previously introduced self-assessed synch-score $s(n)$ (in **??**) 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):

$$errorBuffer = \{\epsilon(n), \epsilon(n-1), ..., \epsilon(n-m)\}, \tag{4.1}$$

then leading to:

$$\begin{aligned} s(n) &= median(errorBuffer) \\ &= median(\{\epsilon(n), \epsilon(n-1), ..., \epsilon(n-m)\}) \in [0,1], \end{aligned} \tag{4.2}$$

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 **??**) 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.

4.3.2 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) [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 Performance-measure: time until harmonic synchrony is detected

GJØR: [Legg inn gode forklaringer på hvordan du implementerte performance-/synch-measuret ditt (vha. **Synchrony Perf.-measure** reMarkable-notatet, schematics (som skissene mine), matte-uttrykk, og evt. algoritmer)].

The performance-measure will be used 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 for achieving *harmonic synchrony* serve as a blueprint or guide for how to implement our synchrony-/performance-measure. This performance-measure should be able to, during synchronization-simulation, 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 4.5. If a certain amount of simulation-time has gone without the detection of harmonic synchrony occuring, the synchronization simulation-run is still terminated and datapoint still saved, but this time as a 'synchronization-fail'.

The resulting and corresponding performance-scores obtained using this performance-measure will then take values of the simulation-time (in seconds) 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.

My specific implementation of the performance-/synchrony-measure 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 [2]). 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 in Figure 4.6.

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. (Regner kanskje med jeg bør utdype litt mer nøyere her med figur, matte, og evt. algoritme-pseudokode.. Eller hva?)

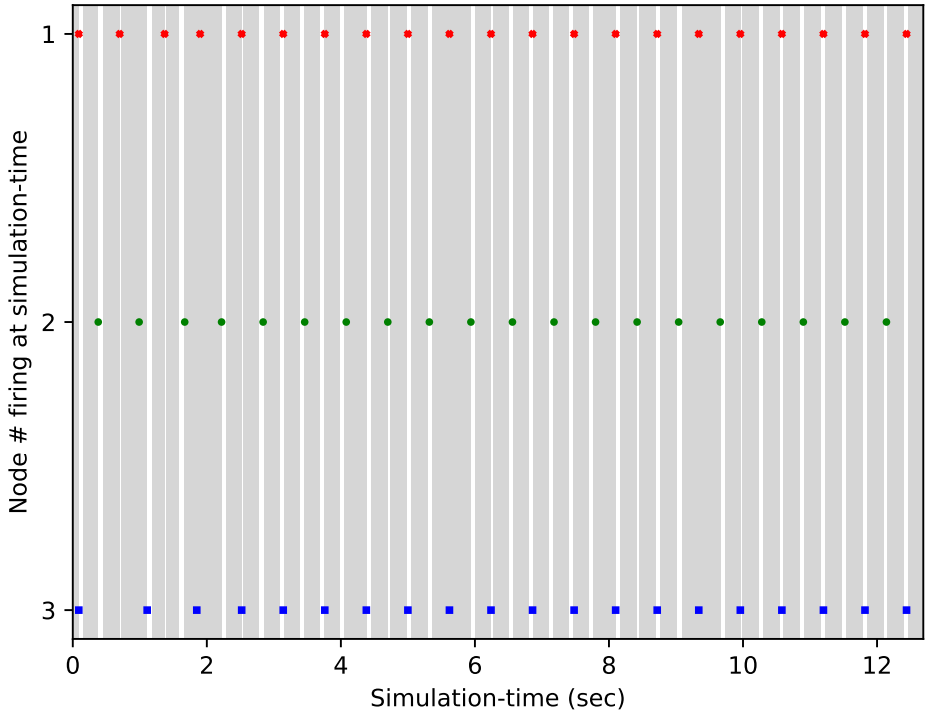


Figure 4.5: A **performance-measure plot**, displaying the temporally recorded pulses/"firings" transmitted by the robots in the Dr. Squiggles-collective throughout the synchronization-simulation in Unity. Short and white windows/strips in the figure represent the short 'legal firing' time-periods t_f during which nodes are allowed to fire within—unless the t_q-duration was just reset (in that case they are $t_f/2$ long, in order to in the future align pulses in the center of the t_f-windows). The larger gray windows represent the 'silent' time-periods within which no nodes are allowed to fire—if the agent-collective is to be harmonically synchronized. In this particular simulation-run above, the robots had to fire evenly $k = 8$ times in a row, within $t_f = 80ms$ long time-windows. As we can observe, harmonic synchrony was eventually achieved after around 12.5 seconds—thereby terminating the simulation-run in Unity as a success (and behind the scenes saving a datapoint consisting of the success-result as well as the 12.5 seconds performance-score, along with the simulator-hyperparameters, to a dataset).

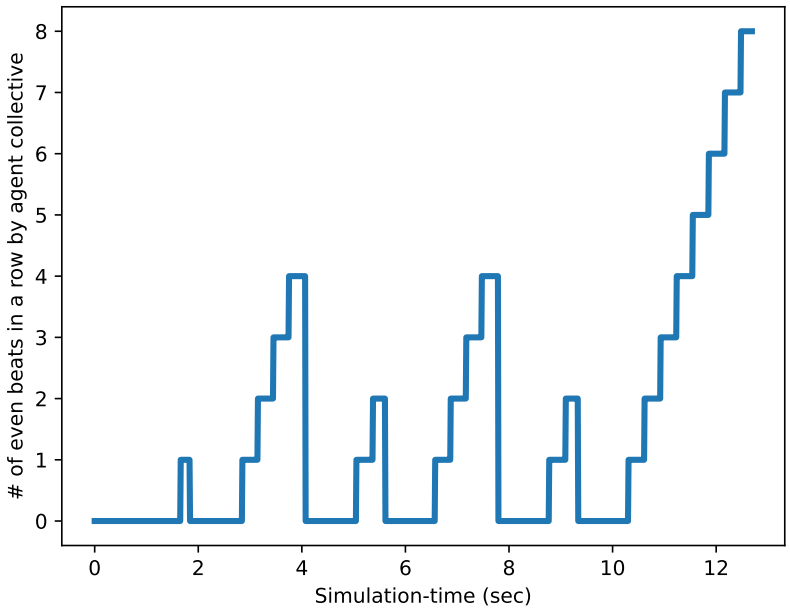


Figure 4.6: A **synchrony-evolution plot**, displaying the temporal recording of the *towards_k_counter*-variable throughout a synchrony simulation-run in Unity. The counter is incremented as the robot-collective fires evenly within 'legal' t_f-windows, and is conversely reset to 0 if illegal firings during 'silent' t_q-windows are heard. Note that in this specific simulation-run above (same run as in Fig. 4.5), the agents were on their way to achieve harmonic synchrony five times before the 10th second 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 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.

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] 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.

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