POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Emergent orchestra: a framework for bio-inspired musical robot swarms

Tesi di Laurea Magistrale in
Music and Acoustic Engineering - Ingegneria dell'informazione

Author: **Pierluigi Tartabini**

Student ID: 10845797
Advisor: Prof. Fabio Antonacci
Co-advisors: Andreagiovanni Reina, Lluc Bono Rosselló
Academic Year: 2025-26

Abstract

Can a group of robots play together without the guidance of a conductor? This thesis addresses this question by proposing a framework based on a decentralized organization of robots, implemented through principles of bio-inspired algorithms and collective decision-making.
The work of this thesis consists to the design and implementation of a software that allows users to configure and run experiments in which a swarm of robots self-organizes to form an orchestral structure. The self-organized ensemble has to respect musical principles configured on the knowledge of each robot, that exploits its logical architecture composed by four modules that operate in parallel, each responsible for a specific musical aspect: rhythm, timing, timbre, and harmony, in order to enable the swarm robotics to achieve coordinated musical behavior. The algorithms developed in the modules are inspired by collective behaviors observed in animal groups, such as fish schools and insect swarms, which show how coherent global patterns can emerge from simple local interactions without the need for a central leader. In addition, the framework draws from collective decision-making theory, where group decisions arise from distributed interactions among individuals rather than from a single controlling agent.

By bridging multiple disciplines, including music, swarm robotics, and collective animal behavior, the proposed framework aims to generate a musical performance whose complexity progressively emerges from the behavioral interactions of the robots, thereby emulating forms of distributed and collective intelligence found in nature. This approach not only enables the production of music, but also offers the audience the opportunity to witness the evolution of the system, because acoustic development makes the principles of collective behavior accessible and tangible.

Finally, the thesis presents a discussion of what could be improved and possible future developments and applications.

Keywords: music, swarm robotics, collective decision-making, orchestra, self-organized, bio-inspired

Abstract in lingua italiana

È possibile che un gruppo di robot suoni insieme senza la guida di un direttore? Questa tesi affronta tale interrogativo proponendo un framework basato su un'organizzazione decentralizzata dei robot, il cui obiettivo è quello di organizzarsi autonomamente per formare un sistema orchestrale.

Il lavoro di questa tesi consiste nella progettazione e implementazione di un software che consente agli utenti di configurare ed eseguire esperimenti in cui uno gruppo di robot si organizza autonomamente per ottenere una configurazione orchestrale. L'orchestra deve rispettare principi musicali definiti nella conoscenza interna di ciascun robot, ognuno dei quali sfrutta un'architettura logica composta da quattro moduli operanti in parallelo per aspetti musicali come ritmo, coesione temporale, distribuzione timbrica e coordinazione armonica. Gli algoritmi sviluppati all'interno dei moduli sono ispirati a comportamenti collettivi osservati in gruppi animali, come banchi di pesci e sciami di insetti, che dimostrano come comportamenti collettivi possano emergere da semplici interazioni locali senza la necessità di un leader centrale. Inoltre, il framework si basa sulla teoria della decisione collettiva, secondo la quale le decisioni di gruppo emergono da interazioni distribuite tra gli individui piuttosto che dal controllo di un singolo agente.

Mettendo in relazione discipline diverse, tra cui musica, robotica e biologia, il framework proposto mira a generare una performance musicale la cui complessità emerge progressivamente dalle interazioni comportamentali dei robot, emulando forme di intelligenza distribuita e collettiva presenti in natura. Tale approccio multidisciplinare non solo consente la produzione di un output musicale, ma offre anche al pubblico la possibilità di osservare l'evoluzione del sistema, poiché lo sviluppo acustico rende i principi del comportamento collettivo accessibili e tangibili.

Parole chiave: musica, orchestra robotica, decisione collettiva, intelligenza colletttiva, sistemi bio-ispirati

Contents

Introduction

This introductory chapter provides a general overview of the context in which the project is situated, focusing on the two main themes of the thesis: music and swarm robotics. Following an initial broad analysis that crosses ideas, motivations and contributions of the thesis, topics are examined from a scientific and computational perspective, in order to equip the reader with the conceptual tools necessary to better understand the subsequent chapters. Finally, a review of related work in the existing literature will be presented, concluding with a discussion of the specific objectives pursued by the project.

0.1. Idea

The thesis consists of the re-implementation and extension of the research framework presented by Bono Rosselló et al. [21]. In the original work, a group of robots with limited computational and communication capabilities were successfully employed to realize a self-organized robotic orchestra. The aim is reached thanks to the communications among the robots of the swarm, that follow collective decision-making principles: these processes are exploited by a group to collaboratively arrive at a decision, making the outcome attributable to the group rather than a single individual [21]. Pursuing this idea, the rule of a central leader (as an orchestra conductor) is substituted by the cooperation among swarm's agents, and none of the robots is necessary to the objective, or none of the robots could achieve alone what the group is capable of. Social consensus does not depend on a single agent of the group; instead, coordination is the result of many interactions and communications between parties. Collective decision-making principles can be observed in our society. For example, in the school assembly, students and staff jointly decide whether to introduce a school uniform policy: each participant contributes individual preferences and arguments, which are discussed and evaluated at the group level. The final outcome, reached through voting or consensus, reflects the aggregation of multiple viewpoints rather than the decision of a single individual.

The original project conducted by Bono Rosselló et al. [21] aimed to exploit principles of collective decision-making to generate a musical output as the result of communications

between robots, each playing a single note. From this point of view, musicians (robots) do not respond to the directives of a central authority, such as an orchestra conductor who sets the tempo to the performers. Instead, the agents self-organize by iteratively shaping their decisions, progressively converging toward shared configurations. Such a coordination is reached by decrypting information from what neighbors played, and choosing the right actions in order to converge on a final common solution.

0.1.1. Motivation

Some of the principles described in the field of collective decision-making can be observed in several animal species that:

- do not live in isolation, but instead organize their behavior in groups or swarms;
- lack an internal hierarchical structure, and therefore do not rely on a central figure to direct the behavior of individual members [20].

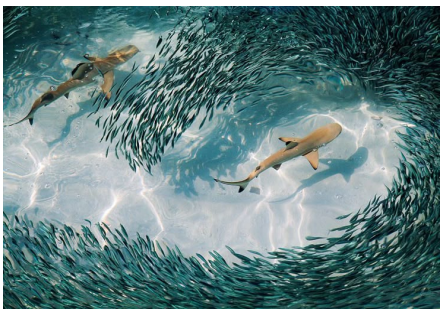
There are plenty of examples in nature about this animal society organization (Figure 1). One example is a school of fish that, when moving in unison, can evade predator attacks [2], or the behavior of emperor penguins that, while incubating their eggs, arrange themselves in a spiral formation to increase the internal temperature (up to 40°C) of the group and rotate positions so that all individuals benefit equally, ensuring that none are left exposed to the cold [6], or even the organized flocking of the birds, in which every individual follows a collective direction [31]. A final common example of collective decentralized organization can be observed in ants which, when faced with gaps too wide to cross individually, connect to one another, forming chains in order to create a bridge with their own bodies and overcome the obstacle [27]. In particular, the analysis of the behavior of fireflies and ants will be peculiar for two of the four modules of the framework.

Can the animal collective behavior enhance swarm robotic performance?
The most important motivation of the thesis is to translate collective decision-making, observed in the animal world, into algorithms that can be employed by robots to perform music without the need for a centralized coordinating figure, and this concept is referred in the title with the **bio-inspired** noun.

These concepts and other biological phenomena form the core research focus of the **Centre for the Advanced Study of Collective Behaviour (CASCB)**, where this thesis was developed under the supervision of Andreagiovanni Reina and Lluc Bono Rosselló from October 2024 to June 2025. The Centre, a cluster of excellence jointly established by the Max Planck Institute of Animal Behavior and the University of Konstanz in Germany,

is dedicated to the investigation of a wide range of collective behaviors across different species and environments; by integrating behavioral, neural and technological perspectives, it adopts a quantitative approach to provide fundamental insights into collective behavior and to support the development of new principles for the design of engineered systems. Thanks to the collaboration between computer scientists and biologists, the project benefited from an interdisciplinary environment and was further enriched through participation in conferences and seminars, which provided valuable ideas and insights.

(a) Ants forming a bridge



(b) School of fish avoiding predator

(c) Emperor penguins forming spiral

(d) Collective flocking of the birds

Figure 1: Examples of collective behaviors in animals and bio-inspired concepts for robotic swarms.

0.1.2. Contribution

In the previous work [21], the swarm reached musical self-organization thanks to three modules implemented on each agent. The parallel work of each module enabled the swarm to reach harmonic and rhythmic agreement, giving a solid starting point to build a swarm robotic orchestra. The contribution of this thesis focuses on two main aspects:

- **Implementation of the simulation framework** for designing and testing new swarm orchestra algorithms. Unlike previous work, where experiments were performed using both real robots and the ARGoS (Advanced Robot Simulator) environment, this framework is designed to be more accessible and less tightly coupled to

robotics-specific details. This choice aims to accelerate experimentation by shifting the focus from the low-level behavior of individual agents to the musical capabilities emerging at the swarm level.
Additionally, the possibility to generate an audio/video file of the resulting simulation has been integrated into the framework.

- **Improving musical quality**: The robotic swarm was able to achieve musical coordination with respect to rhythmic and harmonic aspects; however, the generated audio merely reproduced musical notes without incorporating timbral properties. The second contribution of this thesis is to extend the musical capabilities of individual agents by leveraging collective decision-making principles to improve the quality of the musical output.
- **Implementation of a new musical module**: Finally, the third contribution of this thesis is the creation of new coordination modules. In particular, the work of this thesis has led to the development of the timbre module. Inspired by the collective behavior of ants, which divide into labour castes, the robot form different timbre groups, enabling a more structured musical output. As a result, the robotic orchestral swarm exhibits a richer musical evolution and enhanced expressive features in the produced audio.

While the proposed framework builds upon core principles of the previous work (such as modularity and scalability), it significantly extends its capabilities by introducing a new simulator, a new module, and modifying existing ones, expanding the musical expressiveness of the swarm. In addition, it also enabled the generation of audio/video output from configurable simulations. These contributions are presented in detail in Chapter 1, whereas the implementation details and experimental setup of the original study are examined in Section 0.2.2.

0.2. Foundational Concepts

This paragraph provides an initial overview of the frameworks operating principles. What is essential to understand at this stage is the general structure of the simulation and the action that the robot can implement, which can be conceptualized in the following steps:

1. select an instrument;
2. set the moment on which emit sound;
3. play a note;

4. listen what the other components of the group are playing;
5. select the items in the first 3 points in such a way that the proposed music in the following iteration improves upon the previous one;

It is important to note that the agents are intentionally designed with extremely limited capabilities: the aim is to regulate how musical structures emerge from the individual contributions of the swarm, and not from the abilities of a single one.

The resulting scheme can be interpreted as a simpler case of what is outlined as a jazz performance, where musicians must be ready to respond to a musical proposal made by someone else, ensuring that the resulting flow of the music remains coherent, smooth and enjoyable.
The concept of collaborative creativity is explored in the paper *"Creative collaboration and collaborative creativity: a systematic literature review"* [4], focusing in particular on the jazz tradition. In this musical genre, composition often matches with the act of improvisation, a technique that enables musicians to perform together without relying on a predefined score, and the process makes jazz style a valuable laboratory of real-time collaborative music-making.
This concept is encapsulated in the first two words of the title of the thesis, "Emergent Orchestra," and an examination of these two nouns helps clarify its meaning:

- **Emergent**: the generated music is the result of repeated iterations among the robots; consequently, what is audible is not the product of any premeditated plan by an author or composer, but rather literally "emerges" during its performance thanks to the bio-inspired interactions.
  The emergence of a collective behavior in a robotic context has been studied also from Zhicheng Zheng, Yongjian Zhou, Yalun Xiang, Xiaokang Lei and Xingguang Peng in "Emergence of Collective Behaviors for the Swarm Robotics Through Visual Attention-Based Selective Interaction" [34]. The study presents the *Departing Level of Neighbors (DLN-S)* model, an innovative swarm robotics framework inspired by the predatory behavior of locusts, which eliminates the need for explicit velocity alignment among robots. The research introduces the concept of DLN, a visualbased function that enables individual agents to make motion decisions using exclusively the relative positions of neighboring agents. After experiments with a swarm of 50 robots, the authors demonstrate how selective interactions lead to the spontaneous emergence of complex collective behaviors, such as flocking, milling, and swarming, highlighting its robustness and scalability. The approach reduces hardware computational cost, facilitating the deployment of robotic swarms in complex real-world

environments.

- **Orchestra**: although traditional orchestral groups do not typically employ improvisation, the term "orchestra" is used here to refer to the range of instruments ideally available for the robots, as a music shop, where they can take or change any instrument they want. Improvisation is usually performed with a more limited set of instruments, often referred to as an ensemble. In order to allow the framework to operate across the widest possible range, the term "orchestra" is employed to denote the largest ensemble achievable. A distinction that further clarifies the concept is proposed in the book *"Group Creativity: Music, Theater, and Collaboration"* [24], where the author differentiates between **improvisation** and **structured performance**. Although both represent forms of group creativity, the former is typically associated with smaller ensembles, while the latter is more often linked to larger orchestral settings: both configurations can be emulated from the framework, whose peculiar characteristic is the **scalability** among different robots number, thanks to the application of collective behavior, that are not restricted to the swarm size.
  Another motivation for adopting the term *orchestra* in the title lies in the ability of the framework to reproduce a broad variety of instrumental sounds, reflecting the orchestra as the ensemble characterized by the highest instrumental diversity.

Additionally, the term "orchestra" has been used in the title because it is closely associated with the figure of the conductor, a managerial and organizational role that guides the musicians during a performance. The proposed framework aims to operate without this figure, relying instead on iterative interactions among the individual robots, thereby developing a musical group whose structure is **decentralized** and **self-organized**. Consequently, the characteristics discussed above regarding *collaborative creativity* are crucial in allowing the Emergent Orchestra to function autonomously, without relying on the conductors managerial role. In particular, the embodied concepts that the framework aims to implement are:

- **collectivity**: each musician corresponds to an agent, where no individual one is essential for the task but every singles effort is fundamental for the performance;
- **distributed knowledge**: the collective engagement enables achievements that no single agent could accomplish on its own, and each agent contributes to the emergent outcome through repeated interactions and coordination, highlighting the importance of distributed intelligence and self-organized collaboration.

As can be intuited from the analysis of the first two words of the title, the thesis project has a multidisciplinary nature, and this characteristic is also mentioned in the project

results.

0.2.1. Multi-agent systems

From a software point of view, each robot is figured as **agent**, that can be thought as an element that perceives its environment through sensors and acts on that environment through actuators [13]. The reactivity of an agent depends on its ability to perceive the surrounding environment, while its actions derive from the internal processing of the received inputs. The concept of a reactive agent can be summarized through three key elements:

- **perception**: the information that the agent acquires from the surrounding environment. This can occur either through sensors or through input data provided externally;
- **action**: once the input has been processed, the agent reacts by producing an output in the form of an action;
- **knowledge**: each agent possesses a body of knowledge that may either be *encoded* internally by its designers or *learned* over time through multiple interactions with the environment.

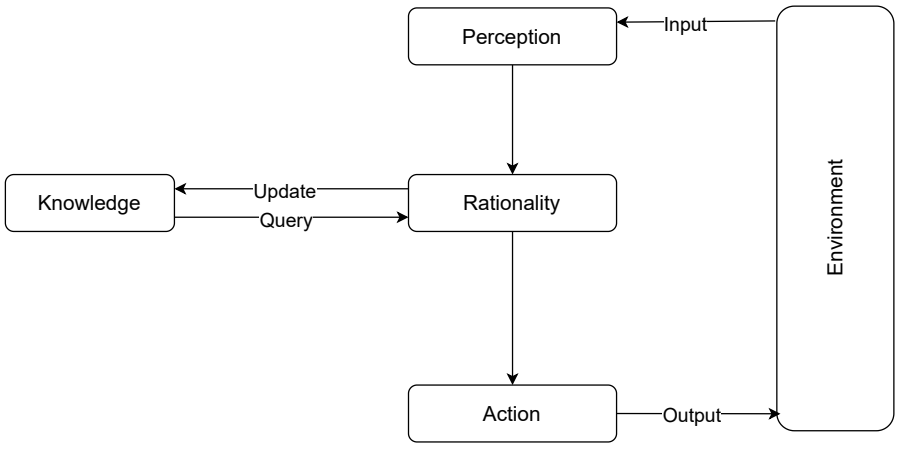


Figure 2: Schematic representation of a reactive agent.

As illustrated in the diagram, an agent performs actions based on the inputs it perceives from the surrounding environment and the knowledge it has acquired. For this reason, such an entity can be described as a **creative agent**, since its behavior may involve actions that are *fixed* (e.g., drawing a straight line), *chaotic* (random behavior), *periodic*

(following a recurring pattern), or *complex* (stochastic but governed by an underlying structure). In the literature, several types of creative agents can be distinguished:

- **cognitive agent (CA)**: maintains an internal symbolic representation of the environment;
- **reactive agent (RA)**: does not rely on an explicit internal representation of the external world;
- **hybrid agent**: combines features of both cognitive and reactive agents.

In the proposed framework every robot is a RA, since they have no prior knowledge of the number of individuals present on the stage. They cannot maintain a symbolic representation of their environment; instead, they respond dynamically to perceived inputs, guided by behavioral rules that derive from their internal knowledge , keeping track of the current state of the world, using an internal model [33]. Robots have the same characteristics of an agent:

- *perception* refers to the ability of each robot to listen in real time to the notes played by the others.
- *knowledge* denotes the internalized set of basic musical rules that the robots rely on to refine and improve the collective musical output; the specific structures underlying this musical knowledge will be examined in greater detail in the paragraph 1.1.6.
- *action* is represented by the capacity of the robot to produce a note selected after processing external inputs and consulting its internal knowledge.

Since the framework is composed of multiple reactive agents, the literature refers to such structures as **Multi-Agent Systems (MAS)**: according to the definition provided by Haopeng Guo, Tao Wu, and Xiang Xu, these systems consist of distributed intelligent agents that collaboratively integrate their capabilities through cooperative interactions, thereby enabling the accomplishment of complex tasks that would otherwise be unattainable for individual agents operating in isolation [11]. MAS can take inspiration from nature, particularly the animal world, as their primary focus is to understand how swarms without a designated leader are able to self-organize in pursuit of a collective goal. A classical illustrative example is that of an ant colony, where each ant can be modeled as a RA, which neither possesses an explicit representation of its surrounding environment nor constructs a symbolic map of it, but it responds to specific inputs through a predefined set of actions. For instance, when an ant leaves the nest in search of food, it deposits a pheromone trail that other ants can follow: If the ant finds food, it returns to the nest

along the same path, reinforcing the trail with additional pheromone; otherwise, the ant returns without depositing anything, and the existing trail gradually evaporates. Some ants may explore alternative paths, introducing fluctuations in the system: this indirect communication mechanism, based on pheromone deposition, illustrates how local interactions among individual ants give rise to a coordinated, adaptive collective behavior that permits to find the optimal path for the food. In this way, rather than stemming from a **single** source of intelligence, a collective **swarm intelligence** emerges, arising from the aggregation of individual behaviors and their interactions within the colony. Whereas the behavior of a single agent was previously characterized as an action resulting from an input and the selection of internal rules, the interaction among multiple agents now gives rise to a **social behavior design**, which can be modeled in various ways [32]:

- **no interaction**: agents do not interact;
- **personal goal**: each agent possesses a goal, is aware of the presence of other agents, and interacts with them in order to achieve its objective;
- **collective goal**: each agent maintains a specific goal, recognizes the presence of other agents, and engages in interactions with them to accomplish its objective.

In the proposed framework, each robot does not pursue a collective goal, but it has an individual objective, namely to play a note that contributes to improving the current musical texture, thereby ensuring that the output of the next iteration enhances the previous one. When each robot pursuing its personal objective, an implicit collective goal emerges: the swarm robotic orchestra gradually improves the music presented to the audience over time. Both in the framework and in the previous example, the collective behavior of the group is emergent because the rules concerning the parts of the swarm do not contain any notion of the whole, and the system is **self-organized**.

0.2.2. Robots in the simulation

The framework is designed to be versatile and can be implemented on a wide range of robotic platforms commonly used in collective robotics research and education. In particular, any robot equipped with basic functionalities such as the ability to produce sound, exchange short messages, and optionally move within a delimited space can support the framework. This includes widely adopted hardware like E-pucks and Thymio robots [18].

The initial implementation of the project was based on real robots specifically designed for the realization of the Emergent Orchestra: robots were disk-shaped units with a diameter of 30 cm, featuring a 3D silhouette of a human head mounted on top (see Fig. 3). They

were equipped with differential drive wheels allowing them to move at a speed of 25 cm/s, loudspeakers for sound production, and infrared transceivers enabling short-range communication (up to 50 cm line of sight). Communication bandwidth was deliberately limited, as the robots could exchange only 1-byte messages twice per second.

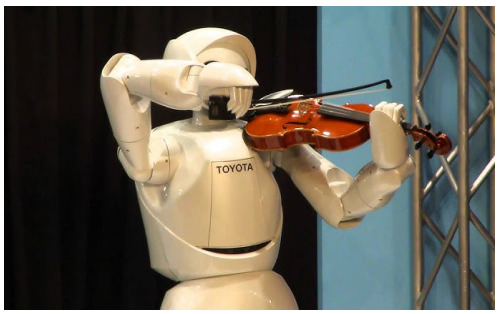
The present framework builds upon this initial implemetation: it began with a replication of the system developed at the Université Libre de Bruxelles and was subsequently extended by introducing additional modules and refining some of the original components. The specific developments and differences will be examined in detail in the following chapter.

Figure 3: Robots used in the first implementation of the Emergent Orchestra at the Université de Namur. A video of the experiment is available at the link https://youtu.be/ZMgT9RWz80.

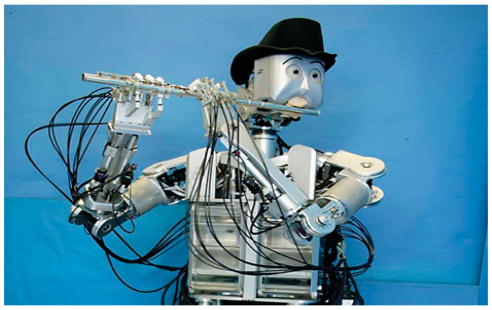
0.3. Related work

Scientific research is increasingly engaging with computational approaches applied to the arts: within this context, music has emerged as a particularly fertile ground for experimentation, especially in robotics: in 2011 Jorge Solis and Kia Ng coined the term *musical robotics* in order to refer to a disciplinary research area involving a wide range of different domains that contribute to its development. The domains include computer science, multimodal interfaces and processing, artificial intelligence, electronics, robotics, and mechatronics [25]. Within the broader domain of musical robotics, different subfields can be distinguished; as outlined by Mason Bretan and Gil Weinberg, *musical mechatronics* refers to the study and construction of physical systems designed for sound generation

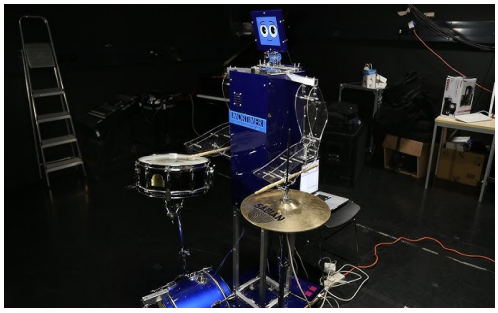
(see Fig. 4): there are many examples of projects about robots that try to play wind [8], string [15] or percussion instruments [12].



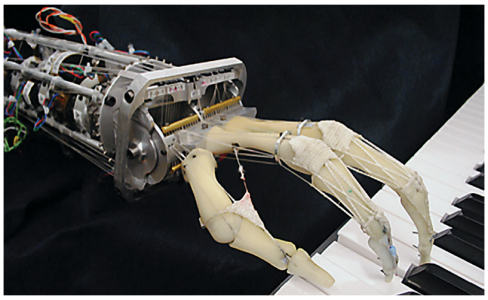
(a) Toyota robot violin player.



(b) Flute robot player from Waseda University.



(c) Shimon, marimba-playing robot player form from Georgia Techs Center Music Technology.



(d) Piano-playing hand from the University of Washington.

Figure 4: Examples of musical mechatronics: robots that play a musical instrument.

Another subfield of musical robotics is the *machine musicianship*, that focuses on the development of algorithms and cognitive models for music perception, composition, and performance not only to imitate human creativity or replace it, but also to supplement it and enrich the musical experience for humans. This area of study is highly attractive to scholars and artists due to its integration of engineering, computation, music and psychology, coupled with complex sensing capabilities using AI to extract high-level musical features from auditory and visual cues. This is the area that comes from the proposed framework. Machine musicianship is fundamentally organized around two primary areas of musical intelligence [7]:

- *Machine perception*: research area that focuses on machines with the ability to sense and reason about the musical world, such as feature extraction, rhythmic and temporal analysis, harmonic and pitch analysis.
- *Generative functions*: involves the capacity of the system to play and perform music, dealing with tasks such as accompaniment, improvisation, and score reading. These functions are based on statistical models, predefined rules, or abstract algorithms.

In the paper "Musical agents: A typology and state of the art towards Musical Metacreation" the authors Kvanç Tatar and Philippe Pasquier listed and reviewed all research projects that employ **musical agents** to creative tasks [26]. The article explores the state of the art of the scientific literature of Computational Creativity, MAS and Artificial Intelligence applied to musical tasks. In the next section will be reported the most similar work to the "swarm robotic orchestra" among the ones mentioned in the article.

0.3.1. Self-organized music

A similar work about self-organized music is presented by Blackwell and Young in *"Self-Organized Music"* (2004), where musical structures are generated through decentralized swarm-based interactions rather than predefined compositional rules [5]. The proposed framework is configured as an artificial improviser, based on the principles of self-organization observed in nature in biological collectives. Through a process called *interpretation*, the system extracts musical parameters from the external sonic environment (human or other input) and translates them into attractors strategically positioned in a Euclidean space, then producing a sonic output in response to the initial input.
The framework consists of two main components:

- **Swarm Music**, a system composed by mini-particles; each of them represents a MIDI note and their positional coordinates are defined by parameters as pitch, duration and loudness. If a musician interacts with the system, music swarm is attracted by notes played by the human and its behavior is to reply with melodic lines related to the input ones, but not identical.
- **Swarm Granulator**, a system composed by mini-particles, where each of them figures a tiny overlapping audio samples fragments called "grains". The Swarm Granulator also reacts to what is heard from the environment, producing as response sound textures and timbres. Sound can vary from irregular to dense flux caused by grains.

Music and Granular Swarm communicate with the *stigmergic mechanism*, a concept introduced in biology to describe indirect coordination in social insects such as ants and termites: in stigmergy, individuals do not communicate directly with one another; instead, coordination emerges through modifications of a shared environment, which subsequently influence the behavior of other agents. Swarm communication allows the emergence of a self-organized musical structure, where the interaction with human performers is mediated by the constant recalibration of attractors in the swarm space.

It can therefore be stated that the main similarities between self-organized music and

robotic orchestra are:

- scalability and modularity provided by the division of musical concepts into structural layers (swarm music and swarm granular in self-organized, the four modules in emergent orchestra);
- the formulation of musical output as a response to events;

The main differences concern the ultimate design goal, as emergent orchestra aims to be used for real robots, while self-organized music aims to interact with external factors.

0.4. Objectives

The aims of this thesis stem from its multidisciplinary nature, which integrates music, robotics, and biology. Accordingly, its objectives include both scientific dissemination and artistic performance.

- **Artistic performance**: the first objective of the thesis is proof that, thanks to the bio-inspired and decision making behavior, robots can execute a musical performance as an orchestra of real musicians, and improve their music level over time; it follows that the final result of the experimentation is a musical performance, in which the robots move within a defined space, resembling a stage. As time progresses, the level of musical texture gradually improves, showcasing the systems ability to evolve in real time.
  A further possible future development could involve an interactive dimension of the performance, in which sounds or music generated by the audience are perceived by the robots and become an integral part of their exhibition.
- **Public outreach**: one of the explicit objectives of this thesis is to promote public engagement with both scientific and artistic concepts through an accessible and immersive musical experience. The resulting performance will differ from traditional concerts, where the audience usually listens to a product they already know, or something that is not too far from the style of the performer or band. In contrast, the performance generated by the robots will always be unique, as the main variables influencing it (e.g., number of robots, beat velocity, time subdivision, and instruments employed) are randomized, making it impossible to reproduce the same outcome twice.
  Another fundamental aspect of public outreach concerns the listener's experience: following the progressive musical development, the audience can grasp basic musical concepts such as harmony and rhythm, and gain insight into the principles

of bio-inspired collective behavior. This occurs because the audience observes a performance shaped not only by musical principles, but also by concepts derived from biology and collective behavior. Consequently, the robots musical performance emerges from the systems progressive self-coordination and stabilization, driven by principles inspired by collective dynamics in the animal world. The nature of the system enables the audience to perceive, through music, the stabilization of bio-inspired interactions among agents and to observe the resulting musical synchronization of the swarm, in accordance with the principles of collective behavior and self-organization discussed above. In summary, the musical performance enables the audience to perceive the emergence and stabilization of bio-inspired collective dynamics, making the process of collective coordination observable through sound.

- **Understanding musical complexity**: listening to each stage of the bio-inspired concert, the audience can learn how different musical layers stratify over time. The nature of the musical performance differs entirely from the conventional one, in which the artist presents a final product that is the result of years of study and refinement. In the performance done by the robots, the audience witnesses the progressive unfolding of both the musical output and the underlying thought process of the robots, step by step: this approach, defined as **bottom-up**, allows the audience to directly observe the evolution of the various musical layers directly from the beginning, thanks to the fact that robots progressively acquire musical knowledge through iterative interactions, whereas human musicians rely on their pre-existing expertise to establish a shared structure as quickly as possible.
  A useful analogy can be drawn with the first music lesson of a drummer or a pianist: during the initial interactions, the musician explores the instrument by striking keys or drums to become familiar with the produced sound; the performance gradually acquire a rhythmic structure only after repeated interactions; similarly, the progressive increase in awareness across all musical layers (rhythmic, harmonic, temporal, and timbral) implemented in the robotic modules is observable by the audience, which can experience the gradual growth in complexity of the performance over time.

1 | System Design and Implementation

This thesis focuses on the development of a framework designed to experiment with the behaviour of a swarm of robots that self-organizes as an autonomous orchestra. By configuring different parameter settings, users can explore how the swarm adapts and coordinates to achieve independent musical organization.
The software was implemented using the *Python* programming language: to run it on a local machine, it is necessary to install a compatible version of Python and configure the corresponding environment variables as described in the official Python documentation .

This chapter provides a comprehensive overview of the softwares functionalities, examining both its algorithmic and logical architecture. It is divided into two main sections: the first describes the structural design of the robots and their modules, while the second investigates the frameworks architecture and underlying the most important steps that occur during simulation. The sequence has been chosen to facilitate a clearer understanding of the design choices and development decisions.

1.1. Characteristics of the robots

Within the simulation, the main agents are the robots, which have the following characteristics:

- **State:** Robots can have two distinct states: **on** (active) and **off** (inactive). Each state persists for a randomly determined duration, introducing variability into the musical performance and enhancing its dynamic character. The state mechanism also models potential real-world conditions such as temporary failures, shutdowns, or malfunctions. By simulating these events, it becomes possible to observe how individual disruptions influence the overall collective behavior of the robotic swarm.
- **Movement**: robots are represented in the framework as two-dimensional circles containing a radius that indicates the current direction of motion, and each of

them is free to move within the delimited stage area implemented through cartesian coordinates x and y. In order to have a similarity with the real world, dimensions of the circles are related to real dimension of robots. To emulate the behavior of the real Thymio robots, each agent is able to avoid collisions both with other robots and with the boundaries of the stage by changing its direction when necessary.

- **Play**: To simulate the role of a musician, each robot can play a musical note at will, using one of the instruments available within the framework. In a real-world implementation, the capability to produce sounds associated with specific instruments will be achieved through the integration of small speakers mounted on each robot.
- **Communication**: to guide the behavior of the swarm in a unified direction, robots must be able to interact with each other: communication among them occurs in a global manner and is designed to emulate the real-time listening process of a musician, perceiving the notes played by the others and extracting meaningful information from those auditory inputs. This exchange of information allows each robot to adapt its behavior dynamically, fostering collective coherence and the emergence of an organized musical performance.

In the real world experiment, Thymio robots are capable of producing sound through the integration of speakers and amplifiers, while communication occurs via infrared signals or Wi-Fi connections. To emulate this functionality, the framework manages inter-robot communication through `supervisor.py`, which simulates the auditory perception of each robot whenever another emits a sound.

In the original version of the project, communication was modeled locally, meaning that two robots could interact only when they were within a certain distance from each other. In the current implementation, however, the framework has been extended to simulate global auditory perception: each robot can hear any note played on the stage, regardless of spatial proximity; this modification enhances the coherence of the simulation, and allows for a more comprehensive representation of musical interaction within the swarm.

1.1.1. Communication among robots

Communication between robots occur in a musical manner: each agent interacts with other components of the swarm playing a note. It follows that interaction manifests precisely at the millisecond when a note is played and is managed by the `supervisor.py` class. The audio-related information that robots can broadcast includes:

- which robot has played (the related *id)*;
- the note that was emitted;
- the instrument used.

The only information not associated with a note is the robots phase, which represents an internal parameter used for rhythmic synchronization within the swarm. To share this value, robots do not rely on global communication; instead, they exchange phase information locally, following the interaction mechanism of the Kuramoto model when neighboring agents fall within a certain distance.
This behavior reflects the use of sensors, exploited by real robot to detect objects and estimate relative distances. However, for simplicity, the sensing range is set to cover the entire movement area, as the design choice allows the framework to prioritize artistic and musical aspects over mechanical and hardware-related constraints.

From an implementation perspective, this information is organized by each robot into an array, which also extends the internal dictionary of that same class to keep track of all notes played throughout the simulation.

The input information acquired by the robots enables the system to evolve in a decentralized manner, as each agent uses it to address the following questions:

- When should I play my note?
- Which note should I play?
- At what point of the measure are we?
- Which instrument should I use to play my note?

In the software architecture, each of these questions corresponds to a specific module, which will be analyzed in the following sections: each module operates synchronously with the others within every individual robot, enabling the entire swarm to achieve rhythmic, harmonic, pulsing and instruments distribution coordination in the shortest possible time.

The framework allows the user to choose which modules the robots will use via the configuration file, which will be explained in detail in the dedicated section.
The ability to specify which modules to activate makes the framework scalable, as it allows users to focus on specific forms of swarm coordination.

1.1.2. Phase Module

In addition to addressing the question **"When should I play the note?"**, *Beat Phase Module* has the aim to achieve a rhythmic coordination of the swarm. From an implementation point of view, every individual owns an internal phase, which functions as an internal clock: once this phase completes a full cycle, the agent is triggered to play a note. The initial value of each phase of the robot is randomly assigned, in order to introduce variability and increase the overall complexity of the simulation.

Rhythmic rules governing the swarm are examined in scripts section concerning agent knowledge and, in terms of architecture, each robot associates an internal phase with every beat that constitutes the musical rhythm. To progress from one beat to the next, the internal clock corresponding to that beat must complete a full cycle and, once the final beat of the measure is reached, the robot cyclically resets the count, starting again from the first beat of the new measure (see Fig. 1.1).

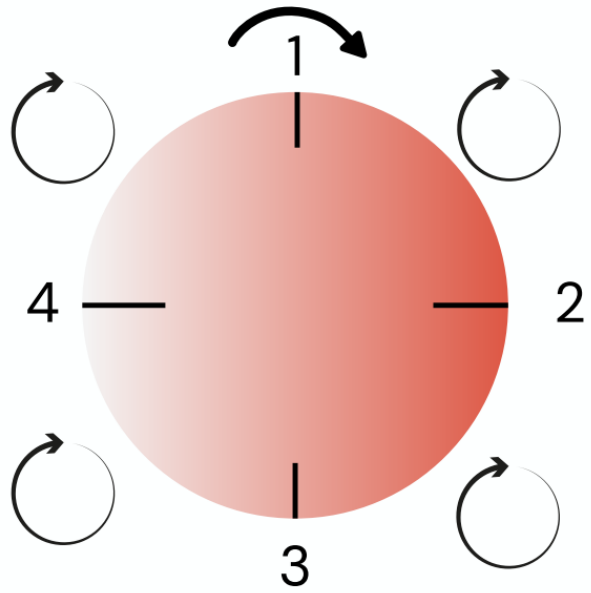


Figure 1.1: Example of internal phase functioning, where numbers correspond to the related beat number and the rounded arrows to the internal phases of the beat.

The image provides a visual representation of the phase mechanism operating within each agent of the swarm: in the example of a 4/4 time signature, each beat has its own phase and, once a full cycle is completed, the system progresses to the next beat in sequence. Once the final beat is completed, the internal clock resets and starts again from beat 1.

To emulate this process, the framework defines the **bar phase value** and the **beat**

phase value: the first corresponds to the total duration of a measure, while the second determines the duration of each individual beat. The relationship between these two quantities is expressed by the following equation:

$$T_{\text{beat}} = \frac{T_{\text{bar}}}{N_{\text{beats}}} \tag{1.1}$$

where:

- T_{beat} represents the *beat phase value*;
- T_{bar} represents the *bar phase value*;
- N_{beats} indicates the *number of beats per measure*.

The computation of the T_{bar} value is handled by the `supervisor.py` class, which extracts the necessary information on the BPM and time signature from the configuration file. This process emulates the initialization phase of the robots prior to the start of the experiment. The algorithm used for this computation is reported below:

Algorithm 1.1 Computation of the Phase Bar Value

1: **Input:** Time signature numerator n and initial tempo BPM
2: Compute the duration of a single beat:

$$s_{\text{beat}} = \frac{60}{\text{BPM}}$$

3: Extract the number of beats n from the time signature.
4: Compute the total duration of one measure:

$$T_{\text{bar}} = 1000 \times (s_{\text{beat}} \times n)$$

5: **Output:** The phase bar value T_{bar} (in milliseconds)

Another important value is the s_{beat}, which is used by robot class to establish the threshold: this value determines the boundary between a beat and another, specifying when a bar phase cycle has been completed. The threshold value is dynamically adjusted according to the tempo of the simulation: higher BPM values correspond to smaller thresholds, ensuring that the beat transitions occur more frequently, whereas lower BPMs produce larger thresholds, allowing slower and more gradual updates of the rhythmic cycle.

After clarifying the structure of the internal clock of the robots, attention is then directed towards examining how its phase evolves over time through successive increments.

Kuramoto model

To achieve rhythmic coordination within the swarm, robots in the orchestra must synchronize their internal phases, so that their playing notes occur in rhythmic alignment. This behavior has been developed through the **Kuramoto model**, originally introduced by Yoshiki Kuramoto [14]. The algorithm provides a mathematical foundation for studying synchronization in large populations of oscillators that interact through phase coupling: each oscillator, or in this case each robot, possesses its own natural frequency but tends to align its phase with those of its neighbors due to the influence of a coupling term. Despite its conceptual simplicity, the model captures the emergence of collective synchronization, as a transition from disordered individual dynamics to coherent behavior. The Kuramoto model is implemented on the robots using a counter θ expressed in radians, which tracks the elapsed time since the beginning of each bar.

The beat phase is updated in two complementary ways: *internally*, through incremental progression at each iteration, and *externally*, by incorporating the phase value of the robot that has just played. From an implementation perspective, the progression of steps corresponds to a new millisecond of the simulation, ensuring a smooth and continuous evolution of its internal clock. The computed equation is:

$$\theta_i(t+1) = \theta_i(t) + \frac{2\pi}{T} \bmod 2\pi \tag{1.2}$$

where $\theta_i(t+1)$ corresponds to the phase counter at time t+1 and mod is the modulo operator, while denominator T corresponds to the T_{beat} mentioned before.

The second update of the phase arises whenever a robot perceives the note played by one of its peers: to simulate the auditory perception of the note, the `supervisor.py` class transmits a message from the robot that has just played to all other robots in the arena, containing the information listed in 1.1.1. Upon receiving the message, each robot extracts the phase value of the sender and updates its own phase according to the following equation:

$$\theta_i(t) = \big(\theta_i(t) + K \sin[\theta_j(t) - \theta_i(t)]\big) \bmod 2\pi \tag{1.3}$$

where K = 1 is the coupling strength, $\theta_i(t)$ and $\theta_j(t)$ are the phase counters of robots i (receiver) and j (sender), respectively.

Synchronization of the beat phase is a key component of the collective dynamics of the swarm, as it quantifies the degree of rhythmic coherence achieved between robots. Phase

synchronization plays a crucial role in determining the temporal precision with which each agent produces its musical output within the measure.

A visual example illustrating the effect of the Kuramoto model can be found at the following link: *Kuramoto Model Visualization*.

1.1.3. Beat Module

The Beat Module, together with the Phase Module, governs the rhythmic coordination of the swarm and aims to synchronize the beat counters of all musical agents. Referring to the 4 questions reported in 1.2.1, this module answers to the doubt **"At what point of the measure are we?**

Each robot has an internal beat counter linked to its beat phase, which increments or resets to 1 at the completion of each beat or measure cycle. Initially, this value is randomly assigned to each robot; while the Phase Module focuses on aligning the mathematical phase values among the agents, the Beat Module is responsible for synchronizing the specific beat position within the measure on which each robot is currently operating.
The Beat Module also relies on communication between robots: whenever a robot plays a note, it simultaneously transmits the characteristics of that note to the rest of the swarm. A **delay** parameter is assigned to each robot, corresponding to an integer value ranging from 1 to N (the total number of beats), which determines the specific beat on which the robot should perform its note; this value is randomly initialized for each agent to ensure a random distribution of beats across the swarm, thereby maximizing the temporal diversity of the musical events perceived by the robots. Figure 1.2 shows an example in which the delay is assigned to the second beat: the robot begins playing once the phase cycle of beat 1 is completed, and the resulting note extends for the full duration of beat two.

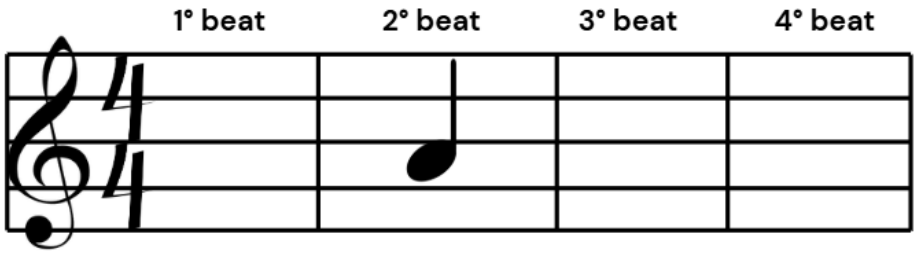


Figure 1.2: 4/4 time signature, 60 BPM, delay = 2.

When an agent performs, the rest of the swarm listens what has been played and extract, from a musical point of view, some related features. The exploited characteristic to

permit beat synchronization is the **dynamic** with which the note has been played, that corresponds on how much air volume does the musician has to apply to that note.

Dynamic enables musicians to diversify the intensity of the note and, in the proposed framework, robots can play their range of notes applying one of these dynamics, mentioned in the table below (see Tab. 1.1) from the weaker to the loudest one:

Notation	Name	Characteristics
pp	Pianissimo	Very soft sound; low sound pressure and minimal intensity.
mf	Mezzoforte	Moderate sound pressure; balanced and natural expressive level.
ff	Fortissimo	Very loud sound; high amplitude and strong expressive character.

Table 1.1: Musical dynamic levels implemented in the framework.

Physics of sound

From an acoustical perspective, a musical note corresponds to a sound wave that carries both kinetic and potential energy. The energy content of a sound wave per unit volume is referred to as *energy density*, often expressed through the concept of *sound intensity* that can be traduced from a musical perspective with the implemented dynamics [16]. Sound intensity can be represented as a vector oriented in the direction of wave propagation, and is commonly quantified by measuring fluctuations in sound pressure, expressed in decibel (dB). For human hearing, the upper limit of perception is known as the *threshold of pain*, and is approximately 20 Pascal (Pa). However, it is more practical to express the strength of a sound by using the logarithmic measure of the sound pressure level, expressed in decibel (dB):

$$L = 20 \log_{10} \left(\frac{\tilde{p}}{p_b} \right) \text{ dB}$$

where $\tilde{p}$ denotes the root mean square pressure, while

$$p_b = 2 \times 10^{-5} \text{ Pa}$$

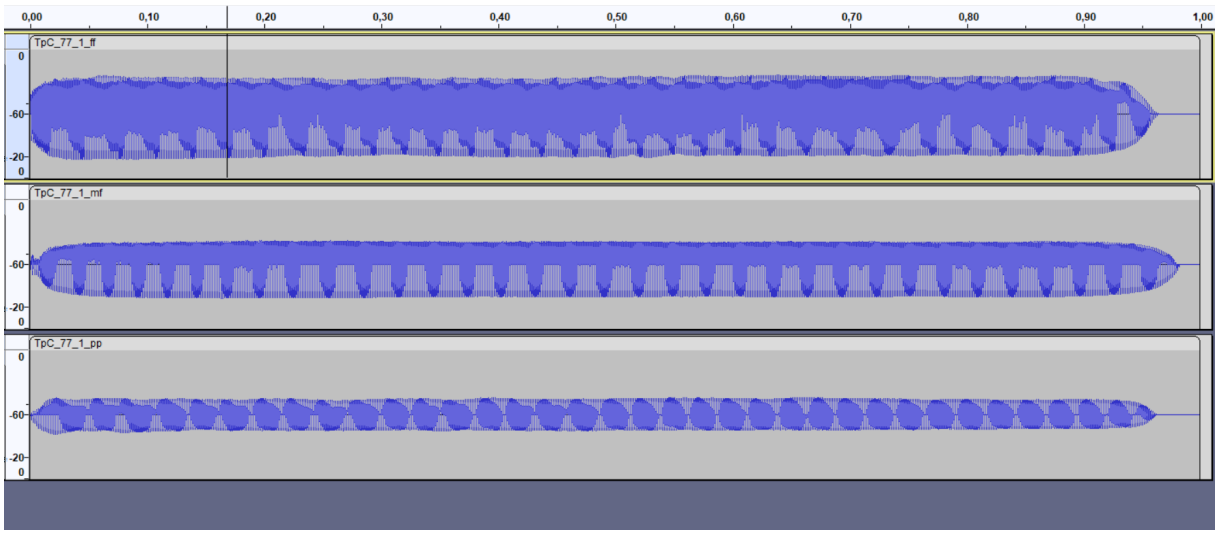


Figure 1.3: Waveform representation of the same musical note at three dynamic levels (*ff*, *mf*, and *pp*).

Figure 1.3 shows the waveform representation of the same musical note performed with three dynamic levels by a trumpet: *fortissimo (ff)*, *mezzo-forte (mf)*, and *pianissimo (pp)*. The amplitude of the waveform increases consistently with the dynamic intensity, reflecting higher sound pressure levels (SPL). In the *ff* recording (top), the waveform exhibits the greatest peak and overall energy, corresponding to an average SPL of **63.80 dB**. The *mf* version (middle) presents a moderate amplitude, with an average SPL of **55.53 dB**, while the *pp* version (bottom) shows a visibly reduced waveform envelope, corresponding to an average SPL of **45.03 dB**. This progressive decrease in amplitude confirms the correct implementation of the dynamic control module in the framework, where the sound intensity varies coherently according to the selected dynamic level.

The dynamic of a note plays a crucial role in the synchronization of beats within the swarm, since it determines the musical accent applied to that note: in order to permit this, the agreement of first strongest beat in western music (described in the internal knowledge section 1.1.6) has been applied. Within the framework, this musical convention is implemented by associating the fortissimo dynamic with the notes performed on the first beat of every measure, corresponding to robots whose delay value is equal to one: as a result, when an agent perceives a note with an *ff* dynamic, it recognizes that the note belongs to the first beat of the measure and accordingly adjusts its internal beat counter, following the synchronization algorithm described in the following section.

Firefly synchronization

The Firefly Synchronization represents the core mechanism of the Beat Module and constitutes the first algorithm within the framework inspired by collective animal behavior. As the name suggests, the method draws inspiration from firefly, a small bioluminescent beetles that emit light from the terminal part of their abdomen.

According to Sarfati et al. (2023) , fireflies do not possess an intrinsic periodicity, because each individual flashes irregularly, with highly variable inter-burst intervals [22]. However, when multiple individuals are within visual proximity, the probability that one of them flashes after the minimal physiological recovery time increases: each flash, in turn, stimulates neighboring individuals to emit their own light, leading the swarm to gradually converge toward a collective rhythmic pattern.

It can be assumed that periodicity in fireflies is not an individual property but an *emergent phenomenon*, arising from mutual interactions and follow the neighbor synchronization, in which any individual can trigger the next collective flash wave. Building on this biological principle, Sarfati developed an agent-based simulation that **extends classical Kuramoto**, and integrate fire models by defining a connectivity coefficient which regulates the strength of interaction among agents and drives the emergence of synchrony (see Fig. 1.4).

Analogously, in the thesis each agent updates its internal beat counter upon perceiving a *fortissimo* note played by another robot, thereby emulating this decentralized biological mechanism of synchronization.
The specific implementation is illustrated by the algorithm:

Algorithm 1.2 Firefly-based Beat Synchronization

Require: Listened note from another agent

1: **If** the dynamic of the listened note is *fortissimo*, set

$$t_{\text{ff}} \leftarrow t_{\text{note}},$$

where t_{note} is the millisecond at which the agent started listening.

2: Compute the relative time between the last bar onset and the most recent *fortissimo*:

$$t_{\text{rel}} = t_{\text{ff}} - t_{\text{bar}},$$

which represents the perceived temporal distance from the last known bar onset to the newly detected note.

3: Convert this time difference into beats:

$$b = \frac{t_{\text{rel}}}{\tau_{\text{beat}}}.$$

4: **if** $b \notin \mathbb{Z}$ **then**

5: **return** ▷ Phase is not yet synchronized; beat coordination would be unreliable.

6: **end if**

7: Compute the current beat distance from the first beat:

$$\Delta = b - b_{\text{actual}}.$$

8: Determine the correction direction by splitting the measure in two halves:

9: **if** $\Delta \leq \frac{N}{2}$ **then**

10: move $\leftarrow -1$

11: **else**

12: move $\leftarrow +1$

13: **end if**

14: Update the internal beat counter:

$$\texttt{beat\_counter} \leftarrow \texttt{beat\_counter} + \text{move}.$$

15: **if** $b = N$ **then**

16: `beat_counter` $\leftarrow 1$

17: **end if**

The visualization below illustrates the graphical representation of the synchronization effect (see Fig. 1.4):

- the background is rendered in black;
- each robot is displayed in yellow when it crosses the threshold of the first beat;
- white lines indicate the current direction of movement.

These colors are not those used in the final simulation video but were specifically chosen to highlight the synchronization process and reproduce the visual impression of the nocturnal firefly phenomenon.

The figure presents 24 sequential frames extracted from the simulation video: as can be observed, during the initial iterations the agents cross the first beat in a random manner, since each robot starts from a different phase. After a few iterations, the agents progressively synchronize their behavior, crossing the first beat at the same millisecond. This visual outcome confirms that the robots begin a new measure simultaneously, thus achieving complete rhythmic alignment.

The emergence of synchrony is studied also by Steven Strogatz in *"Sync: The Emerging Science of Spontaneous Order"*, where the author describes the synchronous flashing of fireflies as one of the most striking natural examples of spontaneous collective coordination [1]. In certain regions of Southeast Asia, thousands of male fireflies illuminate riverbanks by blinking in perfect unison, despite the absence of any leader or central organizing mechanism. Strogatz presents this behavior as a paradigmatic case of a more general principle: when many simple oscillators interact through weak and local coupling, they naturally adjust their rhythms to one another and may eventually converge toward a coherent global pattern. It can be deduced that also in this case synchronization arises from decentralized interactions among individuals.
Strogatz further emphasizes that this phenomenon is not limited to fireflies, but represents a universal pattern observable across biological systems (such as cardiac pacemaker cells), physical systems (such as arrays of coupled oscillators), and even social or technological systems, illustrating the remarkable ubiquity of self-organized temporal order.

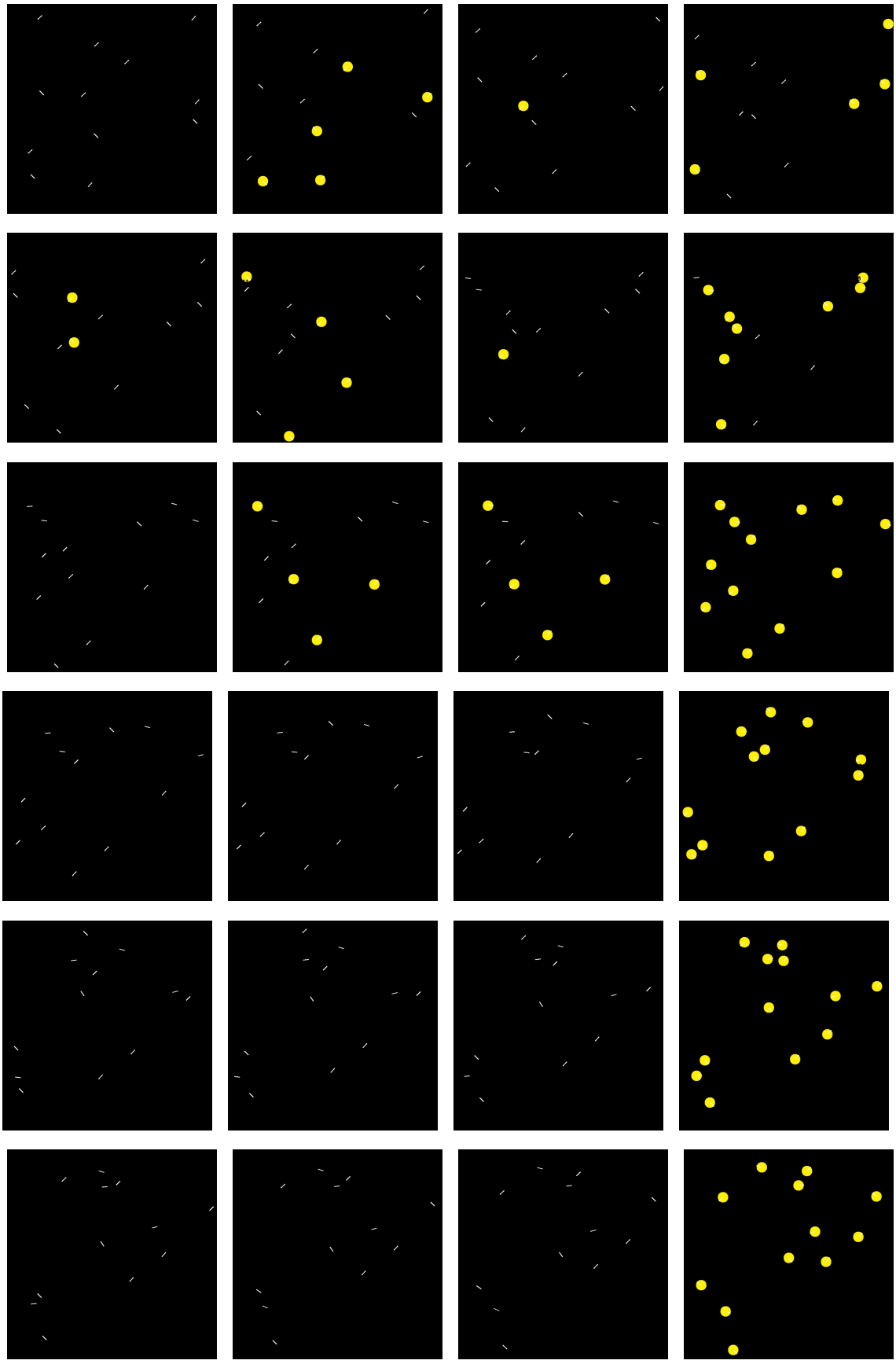


Figure 1.4: Visual representation of the swarm synchronization process across 24 time frames.

In the previous implementation, agents of the swarm shared their beat position to achieve a uniform distribution within the measure: this aspect of the framework has been modified in correlation with the different structure of the internal phase of each robot.

1.1.4. Harmony module

The algorithm responsible for establishing harmonic coherence within the swarm is the harmonic module, which draws conceptual inspiration from the theory of the naming game. This aspect of robotic behavior is grounded in the internal knowledge of each agent, particularly the notion of music scales discussed in Section 1.1.6, with the goal to answer the question **"Which note should I play" ?**
The objective of each robot is to select and perform pitches belonging to the same musical scale, thereby fostering a form of harmonic consensus that emerges through local interactions rather than being imposed by any centralized or hierarchical control structure.
From a more global perspective, the goal of play harmonic note with other individuals can be compared to many aspects of the society that emerges without constraints or rules, as for example dress code, money or also language: all of these categories require *social consensus*, because they are not related to specific commands, but emerges from ordinary common behavior.

How does social consensus emerge from an initially disordered state?
Across the social sciences, this question is addressed by examining analogous processes observed in disciplines such as biology, physics or also artificial intelligence. In these fields, the emergence of consensus is typically modeled as a **cooperative process** in the space of interacting individuals, who progressively coordinate their behaviors and, as a **competitive process**, discards progressively options in the space of available alternatives, because certain options become dominant than others through after agent interactions.

To a better comprehension, A.Baronchielli coined a terminology in his paper [3], and some of the terms are:

- *Population*: set of agents that are involved in the society;
- *Contagion*: transmission of an idea or behavior from an agent to another after communication or contact;
- *Self-organization:*: the ability of a system to develop functional, spatial, or temporal structure in the absence of external control.

These parameters are part of a process that simulates a disordered initial status and, applying a contagion, examines evolution of the group. Baronchielli reported **naming**

game, a process that studies the emergence of a consensus within a population. The naming game process is explained with an example taken from the paper (see Fig. 1.5): an agent may occupy one of three inventory states holding only A, holding only B, or holding both A and B. When an agent holding only A interacts with an agent holding both A and B, the number of agents committed exclusively to A increases. An analogous transition applies to convention B; consequently, the larger the fraction n_A (or n_B) of individuals committed to a convention, the more likely that the convention will spread.

Let n_A and n_B denote the fractions of agents holding only A or only B, and let $n_{AB} = 1 - n_A - n_B$. The evolution of the difference $n_A - n_B$ satisfies:

$$\frac{d}{dt}(n_A - n_B) \propto (n_A - n_B) \tag{1.4}$$

which implies that, in large populations, the convention with even a slight initial majority will eventually dominate.

When the number of admissible states is not restricted to a small number, the naming–game dynamics are characterized by an initial phase of competition among multiple conventions, followed by a regime referred to the most frequent convention that progressively eliminates its competitors. In terms of convergence speed, consensus is reached more rapidly than in classical evolutionary models: the expected time scales as $\log N$ in the binary case and approximately $\sqrt{N}$ when inventories are unrestricted.

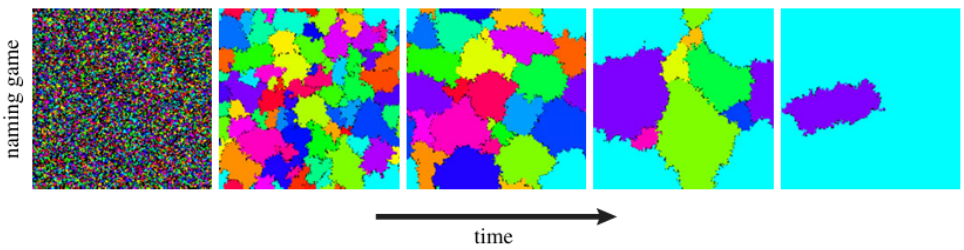


Figure 1.5: Visual example of Naming Game on a two-dimensional lattice, with population of N = 40 000 agents, initial condition with M = N different states: colors correspond to different states. Figure taken from [3].

Naming game in swarm robotics

The `naming game` has been applied for emergence of cosensus in a Multi-Robot network from Vito Trianni, Daniele De Simone, Andreagiovanni Reina and Andrea Baronchelli, where *kilorobots* have been used to simulate a *listener* and a *reader* [30]. In this research, a speaker a_s broadcasts to the population a word w randomly from its inventory, or

inventing a new one if no one are available; once received w, the listener a_l updates its inventory storing the word and, after many interactions, the systems is self-organized converging toward the selection of a single word shared by all agents.

The system inspired the harmony module of the proposed framework, where the values w are replaced by musical notes. In this module, each robot broadcasts a note whenever it plays, and stores the received notes in a buffer of fixed length $L = 4$. Agents record the *pitch* information of each received note and, once the buffer is full, the oldest entry is removed following a FIFO (first-in, first-out) policy, to ensure that each agent retains only the four most recently played notes within the swarm.
The core functionality of the module consists of comparing the currently played note with those stored in the buffer: if all buffered notes belong to at least one common musical scale, the swarm reached an harmonic convergence; otherwise, the agent modifies its note.

To perform this evaluation, each robot inspects every note of all considered musical scales and counts how many buffered notes are compatible with each scale. The scale achieving the highest number of matches is identified as the best candidate, denoted by S_b, and the agent then verifies whether the most recently played note belongs to this hypothesized best scale. If the condition is verified, the robot retains the same note in the subsequent iteration; otherwise, it selects a different note for the next simulation step. This procedure is repeated at every iteration, regardless of whether the previous check was positive.

The change of notes occurs with a probability $r_c < 1$, which is introduced to avoid continuous note replacement that would hinder the swarms ability to reach harmonicity (in the present framework, $r_c = 0.7$). Although agents have knowledge of multiple musical scales, only one scale pattern is instantiated per simulation, enabling an evaluation of the different types of chords and melodic structures that may emerge within the swarm.

Whenever the currently played note does not fit the buffered ones and the probability condition is satisfied (i.e., below r_c), the robot relies on the previously determined S_b with these steps:

- Computes the distance between the last played note and each note belonging to S_b.
- Selects the note with the minimal distance and moves its note by one in that direction, thereby ensuring the smallest possible musical interval between consecutive notes.

Although the note exchange mechanism considers only pitch information, it necessarily affects the corresponding *midinote* values. When the played note lies near the boundaries of the instrument's playable range (as reported in Table 1.8), the robot adjusts the *octave*

of the note in order to preserve the pitch class while selecting the closest valid midinote.

In the next image an example of *harmonic module* is reported (see Fig. 1.6), where different colors correspond to different robots. In the first iteration (marked as (a)) the robot in blue plays the note G because, by listening to F# and C#, it infers that the *social consensus* scale is D, composed of D, E, F#, G, A and B. However, the other robot plays D#, which does not belong to the D major scale. Consequently, in the next iteration (b), after a successful random probability event, the robot in blue changes G to G# with a movement of about a semitone S, in order to belong to the same major scale as the other components, namely B major, composed of B, C#, D#, E, F#, G#, and A#.

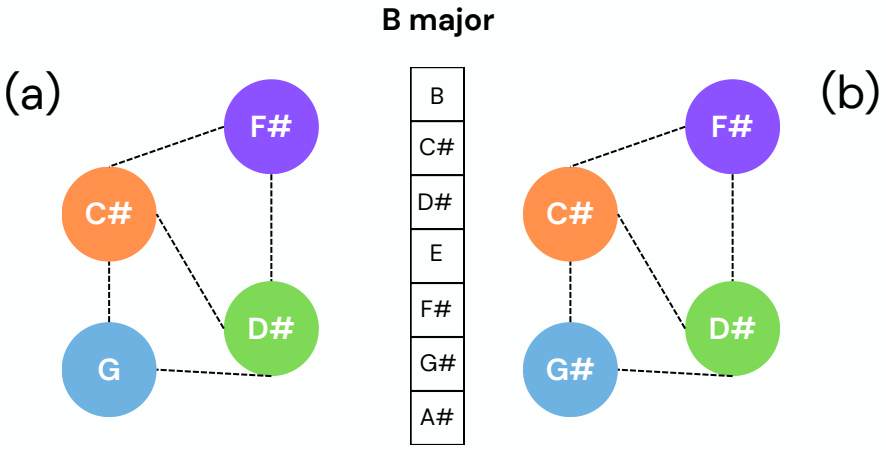


Figure 1.6: Example of naming game application to change the note for harmonic module.

This module was already present in the previous implementation with major scales, what has been modified is the number of possible scales to perform and the relation with notes and available ranges per instrument as reported in the section 1.1.6.

1.1.5. Timbre Module

The central aspect of the project is the sound produced by the robots, which arises from two fundamental musical elements: the note played by the robot and the instrument selected to reproduce it. This paragraph discusses how each agent selects its instrument, in order to contribute the *timbre synchronization* and to answer the question **"Which instrument do I have to choose"?**

Agents can play notes using 13 different musical instruments taken from the sample dataset 1.3 provided within the framework; to allow the swarm to explore a broader

range of configurations, the term "orchestra" has been included in the title: as introduced in Section 1.1.6, modern orchestras can incorporate different kinds of instruments depending on the musical genre being performed; for this reason, non-standard orchestral instruments (such as saxophone and accordion) are also implemented in the project.

Each robot exploits its internal knowledge related to musical ensembles in order to reach a defined configuration, depending on the number of neighbors it can hear. The specific instrument required to match the target ensemble is reported in Table 1.5, and belongs to the internal knowledge of the robot.

In order to achieve a musical ensemble self-organization, a research about bio-inspired algorithms has been done during the internship. In particular, thanks to the help of people that work in CASCB and to the attendance of seminars about collective animal behavior, many algorithms have been examined and taken into account. What emerged from these talks is the animal point of view: in a self-organized society, every action is seen as a **task**, and the objective is to organize the swarm in order to cover all the tasks with the best individuals configuration possible. The assignment of a specific role to an agent is referred to as `task allocation`. It emerges from the continuous exchange of information among swarm members about current system needs, and is therefore considered `dynamic`, as rapidly changing environmental conditions continuously modify task requirements.
The continuous communication among individuals develops a *swarm intelligence*: this skill is fundamental for the animal world to adapt quickly, to react efficiently to any disturbances such as a predator attack. Dynamic task allocation can be seen mostly in insect societies, like for example bees, wasps and also ants: as introduced in the first part, disciplines like *computer science* and *robotics* are studying collective behavior mechanisms to reproduce them for research or industry purposes, in particular to solve *metaheuristic* problems. [1].

Jointly with task, the concept of stimulus is one of the key elements of the **response threshold reinforcement and division of labor in insect societies**, the model developed by Guy Theraulaz, Eric Bonabeau, and Jean-Louis Deneubourg [28]. After some evaluations, this approach has been evaluated as the most fit one with the goal of robots to select the correct instrument, has been implemented for the timbre module applying some modifications. In the next paragraph will be analyzed functionalities and variations made to the original model.

[1] A *metaheuristic* is a high-level optimization strategy designed to guide the search for solutions in complex problems. It provides a general framework that combines simple heuristics to find good-quality solutions without guaranteeing global optimality.

Response threshold reinforcement

Response threshold reinforcement model is applied to implement the decision-making process of the robot, which aim is choose the correct musical instrument, in order to achieve the appropriate ensemble configuration. Response threshold reinforcement operates in parallel with the remaining components, and represents the most biologically inspired component of the project, as it is based on the synchronization behavior observed in fireflies.

The model is based on 3 key concepts:

- **task**: an action that can be performed among range of possibilities.
- **response threshold**: an internal value linked with a specific task, that indicates how much the individual is familiar with that specific labor. These values are defined for each individual and, as suggested by the term *response*, they change according to previous choices. When an individual repeatedly selects the same task, its threshold for that task decreases, as it becomes more accustomed to performing it. Conversely, if the agent does not select a task for many interactions, its threshold for that task increases, since performing it becomes more difficult over time.
  A common biological example can be observed in ant colonies. Foragers leave the nest to search for food, a task they perform frequently and efficiently; when they repeatedly perform the same task, they become increasingly specialized in it. As a result, their response threshold for food gathering decreases; in contrast, ants specialized in nest defense are adapted to guarding the colony. If they switch to a different task, their performance initially decreases because they must adapt to a new activity. For this reason, their response threshold for foraging is much higher, but low for staying into the nest. Each agent has a threshold value for each task present in the simulation.
- **stimuli**: an internal value of the agent that represents the perception of what the population is needing in that moment. Repeating the ant example, if an insect sees that there are a lot of foragers respect to individuals that protect the nest, its internal stimulus to become a nest-ant will increase. On the other side, the stimulus associated to be a forager ant will decrease, because the system does not need it in that moment.

From a musical perspective, tasks corresponds to the available instruments that the robot can play, response thresholds can be seen as
"How good am I in playing that instrument?"

If an individual is able to perform all instruments at a similar intermediate level, the corresponding response thresholds will be approximately equal. In contrast, long-term specialization in a single instrument leads to a lower threshold for that instrument and higher thresholds for the others.
Finally, stimulus can be traduced as
What the swarm needs that I play ?

For instance, if three guitarists are already part of the ensemble and no drummer is present, selecting the guitar in the next iteration would be suboptimal.

Assume that m tasks must be performed, each of them is associated with a specific stimulus whose intensity increases when the task is not sufficiently sustained (either because too few individuals perform it). Let there be N workers, indexed by $i = 1, \dots, N$, each of them has a proper a set of response thresholds θ_{ij} $(j = 1, \dots, m)$ corresponding to the stimuli associated with task j. Let s_j denote the intensity of the stimulus related to task j; in the fixed-threshold model, individual i chooses task j with probability

$$T_{\theta_{ij}}(s_j) = \frac{s_j^2}{s_j^2 + \theta_{ij}^2} \tag{1.5}$$

When $s_j \ll \theta_{ij}$, $T_{\theta_{ij}}(s_j)$ is close to 0, whereas when $s_j \gg \theta_{ij}$, $T_{\theta_{ij}}(s_j)$ approaches 1. At $s_j = \theta_{ij}$, the probability equals 0.5. Therefore, individuals with lower thresholds θ_{ij} are more likely to respond to a given task at lower stimulus intensities.

Additionally, response threshold θ_{ij} of each agent update their values based on task decisions, as mentioned in the description before. Let α and ϕ be the coefficients that describe **learning** and **forgetting** coefficients, respectively. In this time-incremental model, if individual i performs task j by an amount of time $\alpha\,\Delta t$ (that corresponds to the amount of time spent by the individual to perform that task), its response threshold decreases as:

$$\theta_{ij} \;\rightarrow\; \theta_{ij} - \alpha\,\Delta t \tag{1.6}$$

Conversely, if individual i does *not* perform task j within Δt, the threshold increases according to:

$$\theta_{ij} \rightarrow \theta_{ij} + \phi\,\Delta t. \tag{1.7}$$

The two equations can be merged into a unique solution: let x_{ij} denote the fraction of time that individual i spends performing task j; individual i performs task j for a time

$x_{ij}\,\Delta t$, and performs other tasks for a time $(1 - x_{ij})\,\Delta t$. The resulting change in the response threshold θ_{ij} over the interval Δt is therefore given by:

$$\theta_{ij} \to \theta_{ij} - x_{ij}\,\alpha\,\Delta t + (1 - x_{ij})\,\phi\,\Delta t. \tag{1.8}$$

Learning and forgetting coefficients are assumed to be identical for all tasks, and value of θ_{ij} is restricted to an interval $[\theta_{\min}, \theta_{\max}]$.

As for the response thresholds, also the stimuli values of each agent updates every step of the simulation, depending on what modifications happened and what are the requirements for the next iteration. For example, when the number of trumpet players exceeds the required level while violins are underrepresented, the system increases the stimulus associated with the violin and decreases the stimulus associated with the trumpet. The relation that governs stimulus update per unit time is:

$$\partial_t s_j \;=\; \delta \;-\; \frac{\beta}{N}\left(\sum_{i=1}^{N} x_{ij}\right), \tag{1.9}$$

where $\partial_t s_j$ is the derivative of the function over time, δ is the rate at which stimulus intensity increases and β is a scaling factor measuring the efficiency of task performance. Both parameters are assumed to be identical for all tasks, and β is taken to be constant over time and across individuals. The amount of work performed by active individuals (x_{ij}) is normalized by the number of individuals N, to reflect the assumption that demand increases linearly with colony size.

Robots possess internal knowledge of the target configuration they must achieve (see Tab. 1.5). Based on the instruments they perceive, each robot estimates the number of musicians currently present and computes the relationship between the number of players and the number of beats within the measure.
The outcome of this computation is expressed as

$$n_{\text{instruments}} \;=\; \frac{\text{number of players}}{\text{number of beats per measure}} \tag{1.10}$$

which gives the number of instruments that should compose the ensemble.

Computation of number of instruments will generate the related **target distribution** for each instrument: assuming that robot does not listen anyone else in the stage, the target distribution of *Solo* configuration will be only trombone player, so 100% of that

instrument.

The model developed by Guy Theraulaz, Eric Bonabeau, and Jean-Louis Deneubourg was not designed to achieve a predefined target distribution among individuals; its only objective was to ensure that all tasks were performed. The different goal of the proposed framework induced algorithm variation, especially for what concerned the update of stimuli value. In the swarm robotic orchestra, each agent computes dynamically the current **current distribution** based on what instrument they heard: if this distribution does not match the desired final one, it calculates the value of δ as a function of the discrepancy between the current and the target distribution. The δ value is also bounded by a maximal and minimal allowed limit. To resume, the equation 1.9 is applied with different δ as:

$$\delta_j = T_j - C_j \tag{1.11}$$

where T_j denotes the target distribution and C_j denotes the current distribution of the instruments. Indexes are present in the formula because target distribution can change during simulation for robots variable state (on and off) either for increasing amount of musicians during first measure.

Values for parameters implemented in the simulation are:

Parameter	Value
Learning rate (α)	8
Forgetting rate (ϕ)	3
β	3
Minimum δ	0.5
Maximum δ	500
Threshold	500
Stimuli	500

Table 1.2: Parameter values used in the model.

In the next sequence of frames is reported an example of reached target distribution (see Fig. 1.7). In the simulation there are 16 robots for 4 beats and, as reported from equation 1.10, the distribution to reach is a *quartet* made by Trumpet in C, Horn, Trombone, Bassoon. As mentioned in table 1.8, each instrument is represented by a specific color. When a robot crosses the first beat, its color switches to black, allowing visual verification of synchronized arrival at the same millisecond. Red indicates that a robot is playing a note. The instrument distribution is shown in the label: initially random, it progressively converges to the target configuration and remains stable until the end of the experiment.

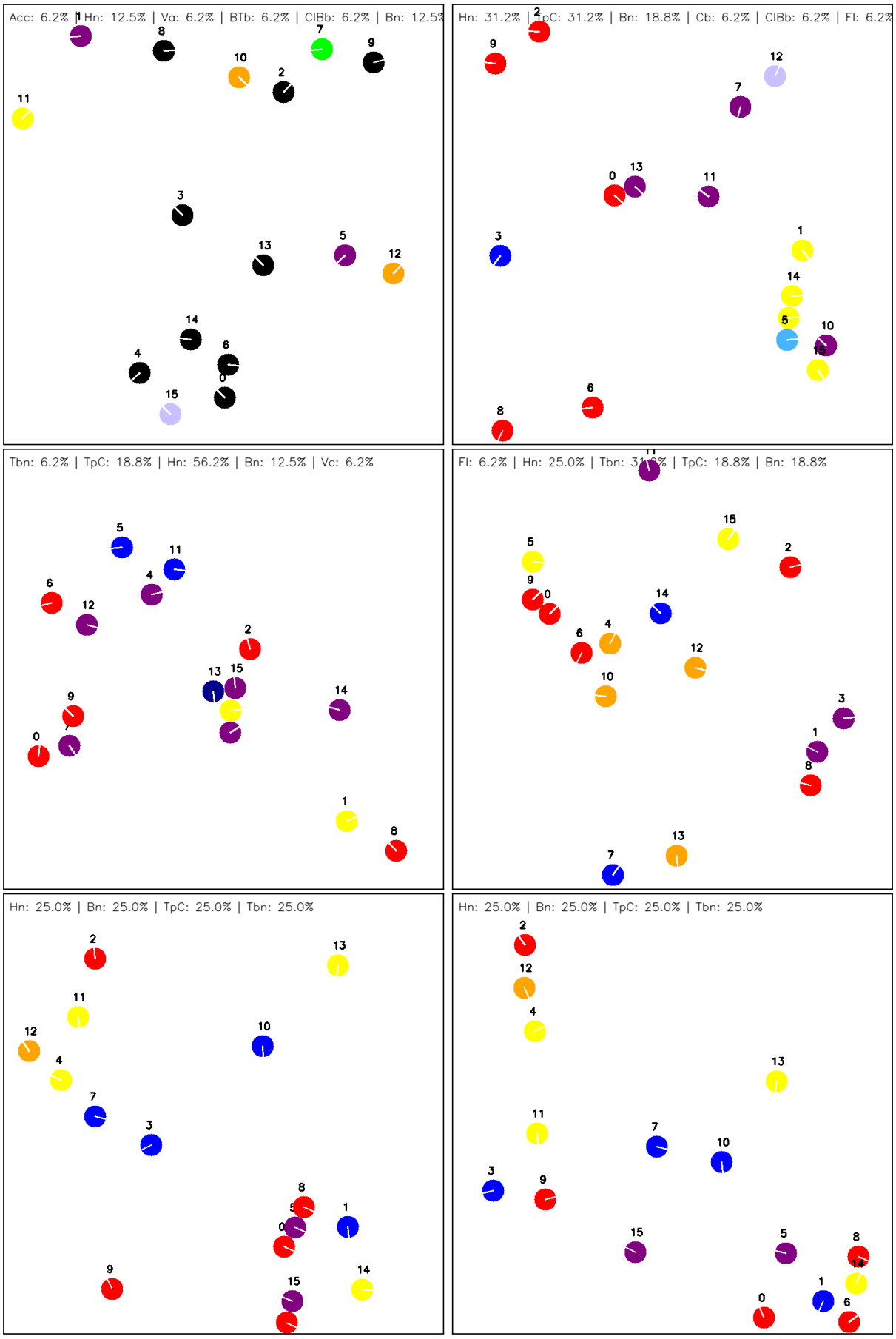


Figure 1.7: Examples of timbre distribution at different simulation times

Another interesting plot is the chronological evolution of threshold and stimuli values of each robot implemented in the simulation. In the plots reported, the simulation had 4 robots that played in 4/4 with a 120 BPM, so they have to play in a *solo* configuration (all of them have to play trombone); δ parameter for this experiment is 100. The legend corresponds to: Violin (blue), Viola (orange), Violoncello (green), Double Bass (red), Flute (purple), Oboe (brown), B Clarinet (pink), Bassoon (gray), C Trumpet (olive), Trombone (cyan), Horn (blue), Bass Tuba (orange), Alto Saxophone (green), and Accordion (red).

In the threshold plots (see Fig. 1.8), it can be deduced that threshold values that not correspond to trombone value tend to increase because they are not needed instruments; sporadically, some threshold decreases their value because that instrument is select for a certain measure. The opposite behavior occurs with regards to the trombone threshold, which decreases its value over time as the robot in question chooses this instrument, respecting the target configuration, and therefore specializing in performing it.

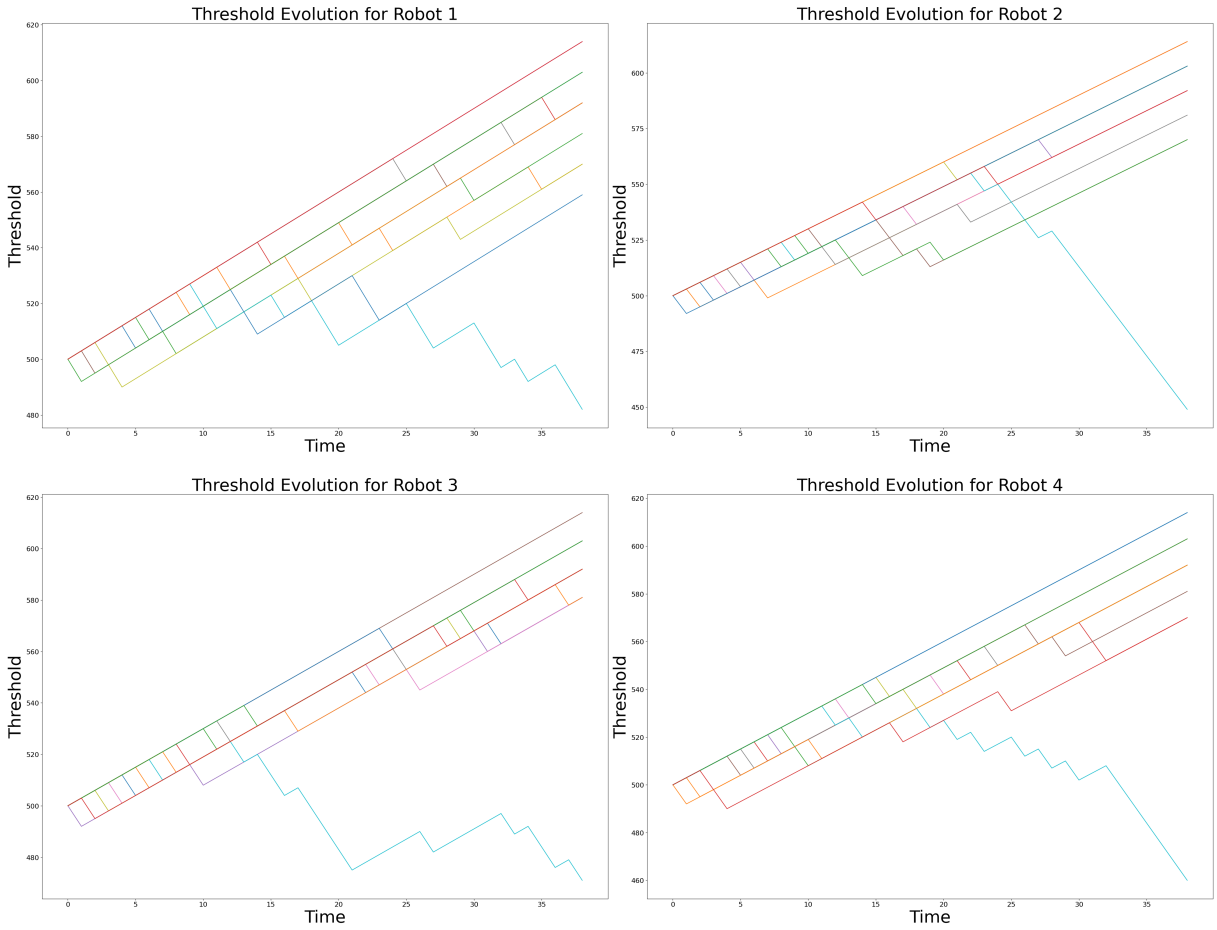


Figure 1.8: Evolution of threshold values for the four robots.

For what concerns stimuli values, the resulted graph (see Fig. 1.9) explicitly states that

only the stimulus to play trombone increases during the simulation, because the configuration to match requires it, while all other values decrease until the minimum possible value. Curves are more or less dense depending on BPMs, because they depend on how many interactions agents have.

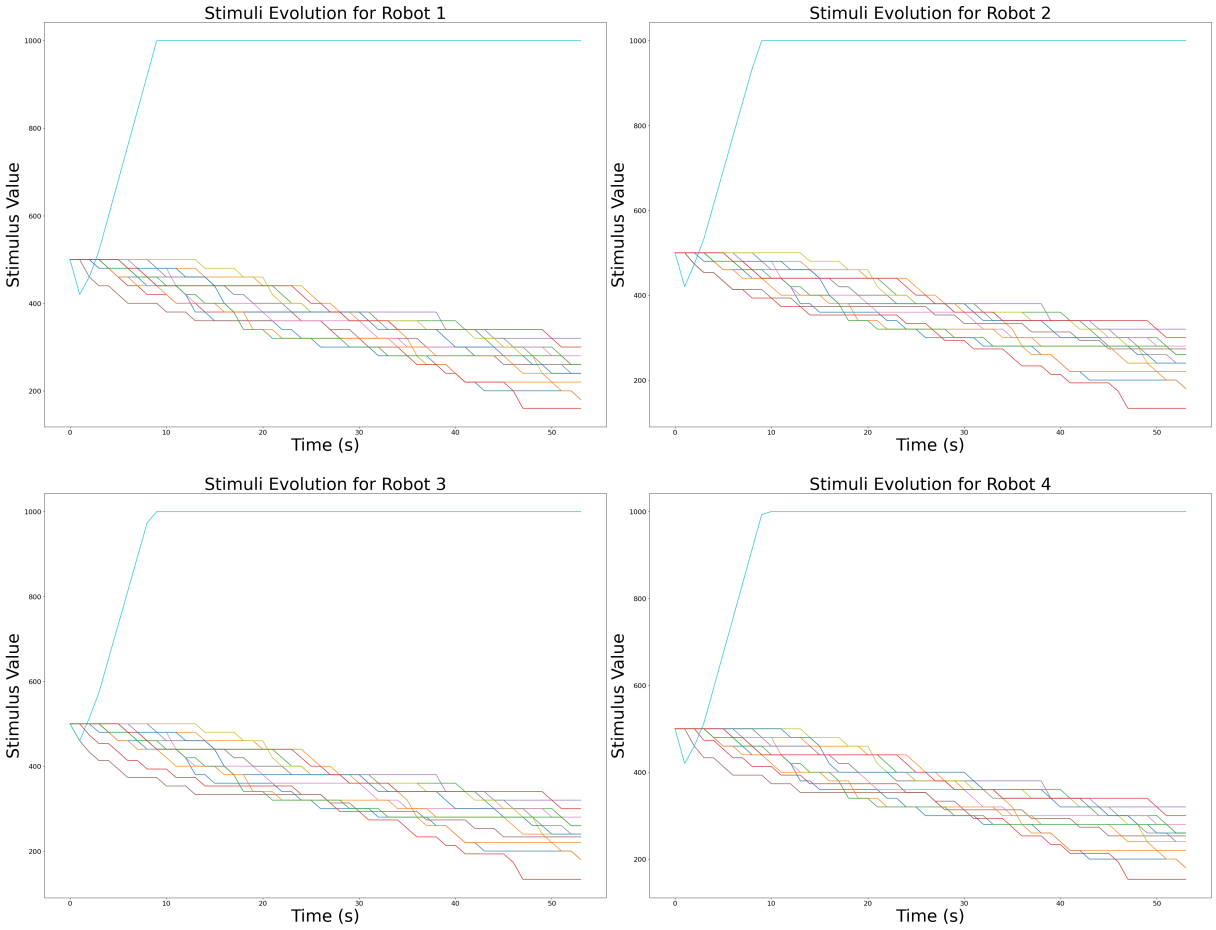


Figure 1.9: Evolution of stimulus values for the four robots.

1.1.6. Internal knowledge

Each agent in the swarm possesses internal knowledge of the musical principles that must be followed to enable coordination across the modules discussed previously. The internal musical knowledge of each agent includes concepts from music theory as well as information about musical ensemble structures. The presence of this internal knowledge characterizes each swarm component as a reactive agent, since it does not possess a global representation of the environment but instead responds according to internal rules. The principles followed by robots regard 4 principal aspects:

(1) Range of notes per instrument

The framework uses a dataset of notes played by instruments: in order to choose the next note to play, robot have internal knowledge of the interval of notes that every instrument can play, in order to avoid selecting a note that that type of instrument cannot play. The note ranges are reported in the summary table in the Database section (Table 1.8), and they are fundamental for the implementation of the Harmony module. If a robot changes pitch and the corresponding MIDI value approaches the lower or upper boundary of its playable range, the robot shifts the note by one octave upward when near the upper limit and downward when near the lower limit, in order to remain within the admissible range.

(2) Music scale

A scale is defined as an ordered succession of eight notes, in which the eighth reproduces the first at the octave. Scales may be classified as *ascending* when they progress from lower to higher pitch, or *descending* when they proceed in the opposite direction [19].

The name of a scale is determined by two components: the first specifies the *tonic*, i.e., the initial pitch of the sequence, while the second indicates the *scale type*, defined by the pattern of intervals applied between consecutive notes. In Western music theory, the fundamental unit of distance between adjacent pitches is the semitone **(S)**, and two semitones constitute a whole tone **(T)**.

Different interval patterns generate distinct scale types, each of which can be constructed starting from any of the twelve pitch classes of the equaltempered system. The interval structures implemented in this framework are summarized in Table 1.3.

Scale Type	Interval Pattern
Major (Diatonic)	T – T – S – T – T – T – S
Minor (Natural)	T – S – T – T – S – T – T
Major Pentatonic	T – T – $\frac{3}{2}$T – T – $\frac{3}{2}$T
Whole-Tone (Hexatonic)	T – T – T – T – T – T

Table 1.3: Interval patterns of the musical scales implemented in the framework (T = whole tone, S = semitone).

To illustrate the process of scale construction, consider the major scale: a major scale is defined by the interval pattern:

$$I = [T,\, T,\, S,\, T,\, T,\, T,\, S] = [2,\, 2,\, 1,\, 2,\, 2,\, 2,\, 1],$$

where intervals are expressed in semitone units.

Given this interval array I, a major scale m built on a pitch class $p_0 \in \{0, \dots, 11\}$ can be constructed through iterative summation of its elements, modulo 12, as formalized in the expression:

$$m_j = \left(p_0 + \sum_{k=0}^{j} I[k] \right) \bmod 12 \qquad \text{with } j \in \mathbb{N},\ 0 \leq j \leq 6. \tag{1.12}$$

The resulting set $m = \{m_0, m_1, \dots, m_6\}$ contains the seven pitch classes of the major scale.

For example, selecting $p_0 = 0$ (corresponding to the pitch class C), the iterative application of the major pattern yields:

$$C \xrightarrow{T} D \xrightarrow{T} E \xrightarrow{S} F \xrightarrow{T} G \xrightarrow{T} A \xrightarrow{T} B \xrightarrow{S} C,$$

producing:

$$\text{C Major} = \{C, D, E, F, G, A, B, C\}.$$

The implemented scale dictionaries based on these principles form the internal harmonic knowledge of each agent and are exploited by the *harmonic module* to guide note selection during the simulation.

To manage musical material within the framework, pitch classes are encoded as integers

in the range 0-11, following the standard modulo 12 representation widely used in computational music theory. In this system, each integer corresponds to one pitch class of the chromatic set, as shown in Table 1.4. This encoding allows scales to be computed through simple modular arithmetic, fully consistent with the construction rule in Eq. (1.1).

Pitch Class Number	Note (American notation)
0	C
1	C# / D
2	D
3	D# / E
4	E
5	F
6	F# / G
7	G
8	G# / A
9	A
10	A# / B
11	B

Table 1.4: Modulo-12 pitch-class representation used in the framework.

(3) Music ensembles

The framework enables agents to self-organize according to criteria that reflect the structure of musical ensembles commonly found in human musical practice. The term *orchestra* in the title therefore refers to an attempt to encompass a wide variety of instruments, enabling the swarm to reconfigure itself into multiple ensemble types depending on the evolving musical context.

In support of this perspective, history shows that musical ensembles have never been static entities, because structures and instrumentation have evolved substantially across different periods. For instance, during the Baroque era (1600–1750), performances typically relied on small ensembles built around the basso continuo and a limited group of melodic instruments, while in the Classical period (1750–1820) the emergence of the symphonic orchestra introduced a more standardized and balanced distribution of instrumental sections (see Fig. 1.10). This development continued throughout the Romantic era (1820-1900), when orchestras expanded substantially in size and expressive range; their internal organization evolved toward a more clearly sectional structure, with instruments

grouped by family, including expanded string sections and a markedly broader range of percussion instruments [10].

Figure 1.10: Example of a symphonic orchestra, in which every color indicates a different section.

Despite the many classifications that exist across musical cultures, modern orchestras generally include twenty or more instrument types, demonstrating the diversity and structural complexity of contemporary ensemble practice. This idea aligns with findings from musicological and computational research showing that musical ensembles and their instrumental roles evolve over time. Mauch *et al.* [17] demonstrate, through a large-scale statistical analysis of more than 17,000 songs from the Billboard Hot 100 [2], that popular music undergoes both gradual changes and "stylistic revolutions" driven in part by transformations in instrumentation and ensemble configuration. Their work highlights how ensemble structures can be understood as emergent, adaptive, and historically contingent.

Within this conceptual framework, the *Emergent Robotic Orchestra* can be interpreted as a computational analogue of such evolutionary dynamics: agents reorganize themselves in response to internal rules and musical constraints, producing ensemble structures that parallel those observed in real musical cultures. This interpretation naturally connects with the modular architecture of the system, in which the timbral, rhythmic, and spatial behaviors of the robots jointly shape the emergent collective musical output.

In the project, agents do not know how many colleagues have, so they apply rules aggre-

[2]The Billboard Hot 100, also known as simply the Hot 100, is the music industry standard record chart in the United States for songs, published weekly by Billboard magazine. Chart rankings are based on sales, online streaming, and radio airplay in the U.S.

gation based in what they listen; in the next table are reported configurations used, that can be modified 1.5. Modalities on which agents of the swarm decide what instrument play are discussed in *timbre module* of robot implementation.

Num. of elements	Name	Instruments
1	Solo	Trombone
2	Duo	Violin, Viola
3	Trio	Flute, Clarinet Bb, Violoncello
4	Quartet	Trumpet C, Horn, Trombone, Bassoon
5	Quintet	Oboe, Flute, Clarinet Bb, Bassoon, Horn
6	Sextet	Violin, Viola, Violoncello, Double Bass, Flute, Oboe
7	Septet	Trumpet C, Trombone, Horn, Bass Tuba, Clarinet Bb, Alto Sax, Violoncello
8	Octet	Violin, Viola, Violoncello, Double Bass, Flute, Oboe, Clarinet Bb, Bassoon
9	Nonet	Trumpet C, Trombone, Horn, Bass Tuba, Flute, Oboe, Clarinet Bb, Bassoon, Accordion
10	Decuplet	Violin, Viola, Violoncello, Double Bass, Flute, Oboe, Clarinet Bb, Bassoon, Trumpet C, Accordion

Table 1.5: Summary of instrument ensembles implemented in the framework.

(4) Rhythm and Metrical accents

In music, rhythm is pronounced by the concept of **beat**, which corresponds to the pulse typically perceived as a non-pitched, percussive sound. The tempo of this pulse is expressed through the **beats per minute (BPM)**, a parameter that quantifies the number of beats that occur in one minute. To maintain a continuous temporal reference, musicians rely on a metronome to count the beats; the metronome can be set to different speeds: higher the BPM corresponds to a faster music.

The beat serves as the fundamental unit that structures time within the performance of the group and, from a musical perspective, defines the **rhythm** that the individuals in the swarm must collectively follow. In traditional music notation, notes are placed on a

staff, and their succession forms the musical piece. For ease of interpretation, the score is divided by vertical bar lines into *measures* (or *bars*) [19], and each measure contains a combination of notes or rests and is rhythmically subdivided into beats, corresponding to the recurring pulses that define the temporal structure of the music. Once a measure is full, the vertical line appears and indicates the beginning of a new one.

Sequence of beats is not sufficient to generate a perceptible rhythmic flow in a musical performance. As Lerdahl and Jackendoff argue in their seminal work [9], beats function as *geometric points* rather than as the *lines* that connect them. To establish the directional flow implied by these points, it is necessary to specify the *meter* of the piece, which reflects the organization of time into recurring patterns of strong and weak beats. The fundamental principle of meter is the **periodic alternation between stronger and weaker pulses**.

The relative strength of a beat depends on its position within the measure: in most Western musical traditions, the **first beat of the measure** is perceived as the strongest, as it provides the listener with a sense of initiation and structural grounding, whereas the second and fourth beats tend to be weaker. A practical example is the *waltz*, in which dancers emphasize their movement every time the music articulates the beginning of a new measure, often resembling a downward gesture.

This hierarchy of metrical accents is not a strict rule but rather a culturally established convention, and altering the placement of the strong beat can give rise to distinct musical styles. For instance, in *blues* and many forms of popular music, emphasis is often shifted to the second and fourth beats (the *backbeat*), creating a heightened sense of propulsion and rhythmic tension, while the first beat becomes comparatively stable and predictable. In written music, metrical stress is represented by the **accent** symbol $>$ placed above a note.

What gives specific rhythm rules for the piece is its time signature, represented as a fraction: the numerator specifies the number of beats contained in a measure, while the denominator defines the note value that represents one beat. A common example is the $\frac{4}{4}$ time signature, which denotes that each measure consists of four beats, each lasting the duration of a quarter note. This is also known as *common time* and represents one of the most widely used rhythmic frameworks in Western music.

The denominator of the time signature plays a crucial role, as it also determines the perceived **tempo** of the piece, and is typically written above the first measure and accompanied by the desired *metronome marking*. In the case of *common time* ($\frac{4}{4}$), if a quarter note is assigned a value of 60 BPM (beats per minute), this means that each beat

corresponds to one second since there are 60 pulses within a minute, one pulse occurs every second.

This temporal unit is conventionally expressed using standard musical notation symbols that represent note durations, allowing musicians to visually associate rhythmic values with their corresponding temporal intervals. In the next table are reported some examples referring as $(\frac{4}{4})$ signature:

Note Symbol	Note Name	Duration
𝅗𝅥	Minima	2 pulses
♩	Semiminima	1 pulse
♪	Croma	0.5 pulse

Table 1.6: Relationship between note symbols and their corresponding duration in pulses.

Within the framework, each robot plays a note that lasts an entire beat (in the $\frac{4}{4}$ example corresponds to a semiminima), while the number of beats contained within each measure is randomly selected in the software from a discrete range of values between 1 and 5. Depending on the BPM value specified in the configuration file, it can perform notes of three different durations.

- 30 BPM, that corresponds to notes about 2 seconds;
- 60 BPM, that corresponds to notes about 1 second;
- 120 BPM, that corresponds to notes about half second;

Rhythms and principle of accented beats are exploited in the section dedicated to the *beat module*.

1.2. Simulator Implementation

After examining the robot implementation and logic in detail, which represents the core component of this thesis, this section introduces the `note.py`, `main.py` and `supervisor.py` classes, which are equally essential for the operation of the framework.

After describing these key classes, the main steps of the simulation are examined to provide an overall view of how the framework operates.

note.py

In the proposed framework, each musical event produced by a robot is represented through an instance of the `note` class, which encapsulates the essential acoustic and symbolic properties of a note. The central attribute is the **MIDI note number**, which provides a discrete and standardized encoding of musical pitch: the MIDI specification assigns an integer $n \in \{0, \dots, 127\}$ to each pitch of the equal-tempered chromatic scale, with the reference value

$$n = 69 \quad \Longleftrightarrow \quad \text{A4} = 440 \text{ Hz}.$$

From an acoustical perspective, the equal–tempered tuning system assumes that the frequency ratio between two adjacent semitones is constant and equal to $2^{1/12}$. As a consequence, transposing a note by 12 semitones (one octave) doubles its frequency, while each individual semitone step scales the frequency by a factor of $2^{1/12}$.

Starting from the reference pitch A4 (MIDI note 69, frequency 440 Hz), the frequency $f(n)$ associated with an arbitrary MIDI note number n is given by:

$$f(n) = 440 \cdot 2^{\frac{n-69}{12}}. \tag{1.13}$$

where the exponent $\frac{n-69}{12}$ counts how many octaves (positive or negative) separate the note n from A4, while the base 2 reflects the octave, doubling relationship in frequency. Each increment of one MIDI unit corresponds to a transposition of one semitone and thus multiplies the frequency by $2^{1/12}$. From the relation, other info can be derived as:

- the **pitch**, which identifies the note name independently of its register (e.g. C, C# , D, . . .);
- the **octave**, obtained through integer division, which determines the register in which the pitch resides.

These two attributes reflect fundamental aspects of musical perception: the pitch value captures the cyclical chromatic organization of tones within the octave, while the octave number reflects the logarithmic structure of pitch, since frequencies separated by one octave maintain a 2:1 ratio.

In addition to pitch information, each note stores:

- a **duration** (`dur`), determined by the BPM and the rhythmic rules defined in the framework;

- a **dynamic level** (`dyn`), associated with one of the expressive categories (pp, mf, ff);
- the **tempo** (`bpm`), which contextualizes the timing of the note.

Through this structure, the `note` class provides a compact yet musically grounded representation, enabling robots to generate acoustically meaningful musical material while supporting symbolic computations within the swarm coordination algorithms.

main.py

The outermost control layer of the simulator is the **main.py**, that governs the temporal evolution of the experiment. As specified in the configuration file, the duration of the simulation is expressed in *milliseconds (ms)*, and each of them corresponds to a single iteration in which every robot executes a predefined set of actions. When a robot is in the *on* state, it performs two primary operations:

- **movement**, implemented along the cartesian x and y dimensions;
- **phase updating**, a process essential for achieving rhythmic synchronization within the swarm and analyzed in depth in the section devoted to the algorithmic behavior of the agents.

supervisor.py

The main module represents the core of the project, as it initiates each swarm iteration. The physical behavior of the robots is simulated by the **supervisor.py** component, one of the principal elements of the framework. This class is responsible for modeling several physical and organizational processes that, in a real robotic system, would be carried out either by human operators or by the robots onboard hardware. The most important setting performance done by the supervisor refers the beginning of the simulation, where each agent has been initialized by assigning:

- state (on/off robot status);
- random initial position within the arena;
- random phase value;
- random musical instrument and a corresponding playable note.

Once configured each agent of the swarm, the simulation can start.

1.2.1. Step 0: initialization

The implementation of the framework begins with the `configuration.ini`, a file that allows the user to define the simulation parameters related to the characteristics of the robots, the environment in which they interact, and several general simulation settings. The customizable parameters are designed to provide the software with a high degree of unpredictability, enabling the generation of diverse types of musical outputs and ensemble configurations.
Specifically, the `configuration.ini` has these parameters:

ROBOT PARAMETERS	
robot_number	Number of robots included in the simulation.
velocity	Movement speed of each robot.
radius	Radius of each robot, represented as a circle with a directional line.
sensor	Perceptual range of the robot, defining its ability to detect other agents.
SIMULATION AND ENVIRONMENT PARAMETERS	
width_arena, **height_arena**	Width and height of the simulation area.
milliseconds	Total duration of the performance in milliseconds.
bpm	Beats per minute, determining the tempo of the musical output.
stimuli_update	Update mode for the timbral stimuli, as described in the following section.
video	Boolean parameter indicating whether a final simulation video should be generated.
number_of_simulations	Number of simulations to be executed.
delta	Delta value that influences the stimuli update of each robot.
modules	Parameter which specifies what modules the robot has to use.

Table 1.7: Configurable parameters defined in the `configuration.ini` file.

The variables refer to two main aspects: characteristics of the robots and the properties

of the stage and music to be generated. Regarding robots, parameters such as *velocity*, *radius*, and *sensor* define their physical properties: although these attributes cannot be modified in real implementations, within the simulation they have an aesthetic and illustrative purpose. The most relevant property that the user can define is the *robot_number*, as it directly influences the type of musical ensemble that the robots will form.

As for the second group of customizable parameters, the most significant are *bpm*, which determines the tempo of the generated music and *video*, which specifies whether or not a final simulation video should be produced.

An important parameter is *modules*, on which user can specify which modules the robot has to activate or not with a string parsed by comma. To activate the modules, user has to type the relative name of the module (as for example *phase, beat, harmony, timbre* for a complete behavior). Customizing this variable, the user can highlight swarm behavior for a specific layer of the music performance, making the framework more flexible and scalable.

Finally, another key parameter is *number_of_simulations*, which allows the user to generate a number of simulations and obtain a dataset to analyzed swarm robotic orchestra under different configurations.
Once the variables are set, the user can start the process from the command line by running `python main.py`, and the parameters provided as input will be processed by the software that, for simplicity, in the text will be called `Simulator`.

1.2.2. Step 1: move and play

The main loop operates at millisecond resolution, where each step defines a potential interaction window for an agent. If a robot has "on" state, it may either move or produce a note when it crosses the assigned beat for sound emission, as described in 1.1.3. When a robot emits a note, the other components of the swarm have to receive the audio input of the colleague, in order to process the information and activate the modules; this principle simulates musician's ears, that are receptive to sounds emitted by colleagues of the orchestra.
The class that emulates robot's ears is the `supervisor`, that manages communication among robots. From an hardware point of view, the mechanism could be implemented adding microphones to robots, while robot's communication would rely on Wi-Fi or infrared signaling. Specifically, **supervisor emulates the auditory mechanism of each robot and the capacity to send and receive messages**, enabling agents to perceive notes played by their peers and extract the associated temporal and musical information.

In the earlier version of the framework, communication followed a *local* scheme: robots exchanged information only when within a certain distance, simulating infrared-based proximity sensing. In the present implementation, communication is modeled as *global*, allowing every agent on stage to perceive any note played by any other, thereby improving decentralized musical coordination. The stored information will be used by each agent to achieve musical synchronization of the swarm, which makes supervisor activity essential for swarm coordination.
An exception concerns the broadcast of the phase value, which operates as a simulated global mechanism. Phase value communication is enabled by setting the distance sensor range to infinity, thereby emulating global communication as for the other parameters.

Once the millisecond time step is completed, supervisor maintains an up-to-date record of all notes played throughout the current iteration for each milliseconds of the simulation: each time a robot performs a note, this class logs the event by storing the played pitch, the instrument used, and the exact millisecond at which it occurred. This mechanism enables the system to keep a complete chronological trace of the musical interaction and, if requested to reconstruct the full audio output of the experiment.
This process is conceptually analogous to the role of a conductor during a live performance: the director follows a master score that integrates the individual parts of all musicians, ensuring temporal coherence and correctness of execution across the ensemble (see Fig. 1.11).

From a software perspective, `supervisor` records all musical events in an array structure named *conductor_spartito*.

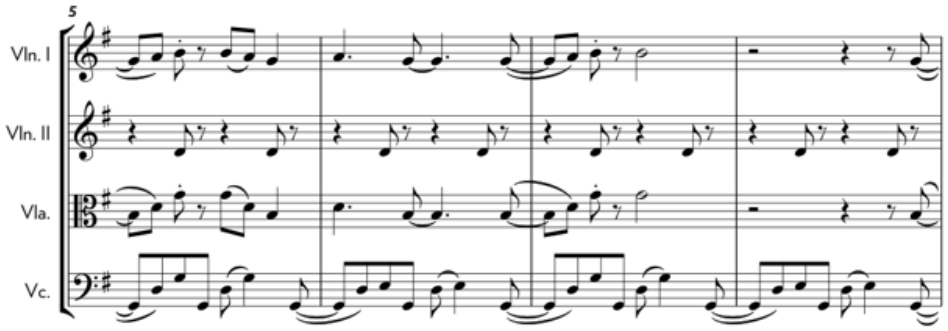


Figure 1.11: Example of a conductor score, formed by the individual musical parts of all performers.

Once simulation has been concluded, the framework builds the final video with the correlated audio of the completed simulation If requested by the user in the configuration

file.

1.2.3. Step 3: final video

The final video of the simulation is an MP4 file made by an *audio* and a *video track*, respectively in FFmpeg and wav format. For a better comprehension, these two fields are analyzed in a separate way, in order to have a better explanation about how the multimedia output is made.

Video track

The script `arena.py` manages all operations related to the creation of the simulation video, and this process is carried out in several stages.

The first task performed is the generation of a dedicated file called `video.csv`, which stores the state of each agent over milliseconds during the simulation. In particular, the following information are recorded:

- simulation_number: index of the simulation run;
- millisecond: current timestamp in milliseconds;
- robot_number: robot's id;
- status: operational state (on/off);
- x: x-coordinate of the agent;
- y: y-coordinate of the agent;
- compass: value used to draw the robots orientation line;
- beat: current beat of the robot;
- phase: current phase value;
- colour: visual color assigned to the robot (each instrument corresponds to a different color);
- *timbre*: musical instrument currently selected by the robot.

The file is written even if the user does not want a final video; consequentially, to avoid excessive memory usage and redundant data storage, these entries are written to the csv file every 40 milliseconds, providing sufficient temporal resolution for video rendering without unnecessary overhead.

To create the final video file, each of the stored milliseconds in the `video.csv` file are converted into frames in the `frames creation` process, that is implemented with the the following operations:

- **Frame initialization**: a blank arena image is created to serve as the canvas for the current frame.
- **Computation of timbre distribution**: the percentage of robots associated with each musical instrument is computed and displayed as a textual label, providing a real-time overview of the ensemble's instrumental configuration.
- **Robot rendering**: each agent present at the current millisecond is drawn on the arena through a call to the `draw_robot()` method.
- **Frame saving**: once the frame is complete, it is exported as a png file and saved in the png folder, contributing to the sequence of images used to generate the final video.

The temporal evolution of the visual scene has been managed producing a frame by frame reconstruction of the swarm's behavior throughout the simulation. Furthermore, the `draw_robot()` method handles the graphical rendering of a single robot within the arena and performs the following conceptual tasks for each agent:

- **Robot body**: the robot is drawn as a filled circle whose radius is defined by the configuration parameters, and whose color corresponds to the robot's current timbre.
- **Orientation line**: a line originating from the robot's position is drawn to indicate its current direction of movement.
- **Robot identifier**: the numeric ID of the robot is displayed near its position, allowing the viewer to distinguish between individual agents.

Through these operations, each robot is visually encoded with information regarding its identity, orientation and musical role, yielding an interpretable and intuitive graphical representation of the swarm.
The final operation performed by arena.py is to concatenate frames in order to produce a video, that merges all png images stored in the rendering directory into a continuous video stream.

Audio track

The final wav file of the simulation is the audio corresponding to the concert performed by the swarm, and its generation is a task performed by `wavGenerator.py` class. Before producing the wav file, wavGenerator writes `music.csv`, a csv file where are stored all the musical data generated during the simulation taken from the *conductor_spartito* of the supervisor. In particular, the most important used to generate wav file of the performance are:

- ms: millisecond on which the note has been played;
- note: note played by the agent;
- dur: duration of the note;
- timbre: instrument used;
- dynamic: intensity applied to the note;

In order to write music.csv, events are logged ordered by ms.
Once the csv file has been created, the second task performed by wavGenerator consists of storing file paths of all the audio samples required for the final rendering within a python dictionary structure: the lookup keys used for this purpose are timbre, note, duration, and dynamic, which directly match the organizational structure of the samples directory, as shown in Figure 1.12.

The final step of the audio pipeline consists in reconstructing the complete waveform of the performance from the symbolic representation stored in the csv file and the set of prerecorded audio samples.
This operation is implemented by the method `generate_audio_from_csv`, which takes as input the list of wav files corresponding to the musical events played during the simulation. The method opens the csv file and reads its last line in order to extract the timestamp, expressed in milliseconds, of the last musical event. This value, denoted as the final millisecond of the performance, is used to determine the duration of the output signal. An additional temporal margin (one second) is added to this duration, and the total length is converted into samples by multiplying by the sampling frequency, which is fixed to 44 100 Hz. A one dimensional array of zeros with this length is then allocated, representing the global audio buffer into which all individual notes will be inserted.

Before processing the events, consistency check is performed by comparing the number of data rows in the csv file with the number of wav files provided as input. The check allows the detection of potential mismatches between the symbolic description of the

performance and the corresponding audio material. Subsequently, the method iterates jointly over the csv rows (excluding the header) and the list of wav files: for each row, it parses the onset time of the note in milliseconds (additional fields such as duration or delay). The onset time is converted into a sample index by:

$$\text{start\_sample} = \frac{\text{ms}}{1000} \cdot f_s,$$

where $f_s = 44100$ Hz is the sampling frequency. The corresponding wav file is then loaded and, if necessary, resampled to the same sampling rate. The audio vector is inserted into the global buffer starting at *start_sample* and extending over its length. Multiple overlapping events are naturally combined by summation of their sample values, thereby implementing a linear superposition of all notes played by the swarm.

At the end of the iteration, the global buffer contains the complete audio representation of the simulated concert; the array is finally written to disk as a wav file, which constitutes the final audio output of the framework and corresponds to the mixture of all individual instrumental contributions generated during the simulation.
The audio and video track will be concatenated in order to export a final MP4 output.

The following algorithm summarizes the main processes occurring during the simulation (see alg. 1.3). All module activations and robot communications originate from the main loop, where time advances in millisecond steps. At each step, a robot may produce a note or receive information from neighboring agents, which it stores for subsequent processing. Meanwhile, the framework records positional and musical data in the `video.csv` file, which is later used to generate the final video, if requested.

Algorithm 1.3 Overall Simulation Workflow

Require: Number of simulations I, number of robots N, parameter Δ, boolean_video

Create and open the main csv file and write the header
for *simulation_number* = 1 to I **do**
    supervisor initializes N robots (states, positions, phases, instruments) using Δ and the time signature
    for each millisecond ms in the simulation time **do**
        for each robot in the swarm **do**
            if robot is on **then**
                robot updates its beat phase
            end if
            if robot plays a note **then**
                Append the note to the global conductor score
                new_note ← true
            end if
            if ms mod 40 = 0 **then**
                if robot is on **then**
                    robot moves
                end if
                Write robot state to `video.csv`
            end if
        end for
        if *new_note* = true **then**
            supervisor broadcasts updated musical information to all robots
        end if
        Reset *new_note* and clear robots' listening buffers
    end for
    Write the global conductor score to `music.csv` for this simulation
end for
if boolean_video = true **then**
    Create final video of the simulation
end if

1.2.4. Project structure

To provide a final overview of the project folder structure, files of the simulator are organized into several key elements: `main.py` represents the sequence of events and iterations throughout the simulation process, while the folder `classes` contains all the Python classes implemented within the project, defining the behavior and properties of

the robots, the environment, and the musical logic. The directory `csv` stores the output data generated during the simulations, including information related to both the musical and visual performances. The folder `samples` is the database of notes that robots can perform. Finally, the folder `plot` collects all the individual frames that are later combined to produce the final simulation video.

The structure of the project folder is shown below:

```
Emergent robotic orchestra/
├── main.py
├── configuration.ini
├── classes/
│   ├── robot.py
│   ├── supervisor.py
│   ├── tempo.py
│   ├── arena.py
│   ├── dictionaries.py
│   ├── signalProcessing.py
│   └── wavGenerator.py
├── csv/
│   └── simulation_number/
│       ├── music.csv
│       └── video.csv
├── samples/
│   └── instrument/
│       └── instrument_note_duration_dynamic.wav
└── plot/
    ├── frame01.png
    ├── frame02.png
    └── ...
```

Figure 1.12: Project folder structure of the simulation framework.

In conclusion, the diagram below presents a comprehensive summary of the operational workflow of the framework (see Fig. 1.13):

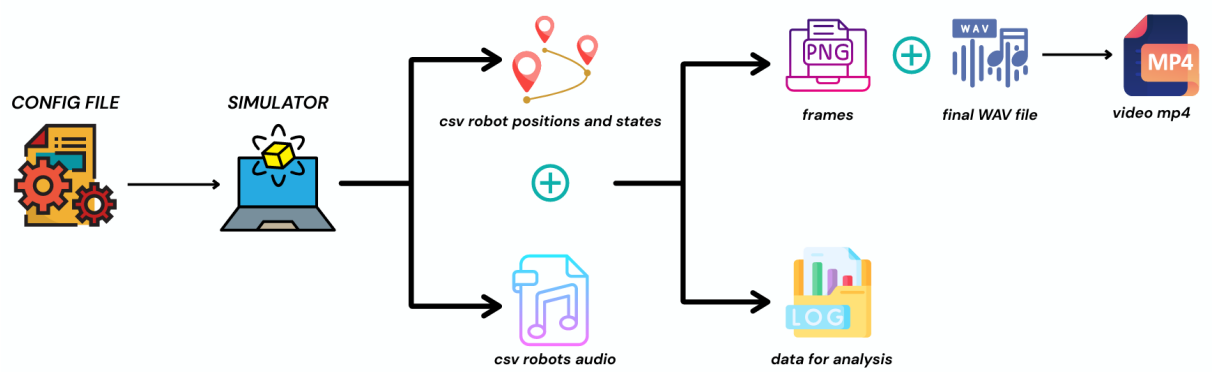


Figure 1.13: General scheme of the framework's operational workflow.

1.3. Database

Once the organizational components of the framework have been examined, this section focuses on the acoustic dimension of the system. In the proposed project, each robot is able to switch between musical instruments with ease unlike human musicians, who are typically trained to perform on only one or a few instruments. This capability is enabled by integrating a dedicated database taken from the TinySOL database from IRCAM [3] and modified to integrate into this framework [29].

The original database TinySOL contained samples of 13 different musical instruments, recorded at IRCAM between 1996 and 1999, and is made by high-quality monophonic samples of orchestral instruments, provided as a 44.1 kHz, 16-bit wav file. The original dataset is organized hierarchically: recordings are grouped first by instrument family (e.g., Strings, Winds, Brass), then by instrument, and finally by playing style. File names encode detailed metadata, including the instrument abbreviation, the played pitch (using American standard pitch notation, such as A4 = 440 Hz), the dynamic level (*pp*, *mf*, *ff*) and additional performance information such as bowing string for string instruments or alternative takes for winds. A final tag indicates whether the recording underwent digital processing, such as tuning correction or pitch transposition, or whether it is preserved in

[3] IRCAM is an internationally recognized research center dedicated to creating new technologies for music. The institute offers a unique experimental environment where composers strive to enlarge their musical experience through the concepts expressed in new technologies.

its natural form. For example, the file

`Strings/Violin/ordinario/Vn-ord-D#7-mf-1c-T22d_R100u`

refers to a violin note in ordinary technique at pitch D#7, played with *mf* dynamics on the first string, synthetically generated by transposing a neighboring pitch upward by 100 cents and subsequently tuned downward by 22 cents to match the standard A440 tuning reference.

Because of this carefully designed structure, TinySOL is widely used not only for creative purposes but also as a benchmark in music information retrieval research, particularly for tasks such as instrument classification and fundamental frequency estimation.

1.3.1. Signal Processing

To integrate the original dataset into the emergent robotic orchestra, several structural and acoustic modifications are introduced: the first set of changes concerns the categorization of the wav files. As illustrated in the directory scheme reported in Figure 1.12, the samples folder now contains one sub–folder per musical instrument, each of which includes all the notes playable by that instrument at three different dynamic levels. Additionally, all pitches contained in the original dataset were converted from american pitch notation (e.g., A4, C5) to their corresponding MIDI numbers: this transformation is performed through a mapping based on the standard equal-tempered tuning system. The resulting MIDI values have been subsequently used to label the processed samples. Implemented instruments in the framework and their range of notes are reported in the table 1.8.

In order to provide a wider range of expressive possibilities, the original note durations were systematically modified: as a result, each instrument sub-folder contains multiple versions of the same note, differing in duration as well as in dynamic level.

Consequently, each wav file follows a standardized naming convention of the form:

`<instrument>_<pitch>_<duration>_<dynamic>.wav`

which allows the system to retrieve the appropriate sound sample during the audio-generation phase.

Category	Instrument	MIDI Range	Samples	Color
Strings	Violin (Vn)	55–100	414	
	Viola (Va)	48–96	441	
	Violoncello (Vc)	36–84	441	
	Double Bass (Cb)	28–72	405	
Woodwinds	Flute (Fl)	59–98	355	
	Oboe (Ob)	58–92	323	
	Clarinet in B^b (ClBb)	52–92	378	
	Bassoon (Bn)	34–75	378	
Brass	Trumpet in C (TpC)	54–87	288	
	Trombone (Tbn)	34–73	345	
	Horn (Hn)	31–77	402	
	Bass Tuba (BTb)	30–65	324	
Other Instruments	Alto Sax (ASax)	49–81	297	
	Accordion (Acc)	28–109	762	

Table 1.8: Instrument families, ranges, number of samples, and corresponding colour markers.

The modification of notes duration has been developed in the script `signalProcessing.py`, which exploits the time scaling technique applied to each wav file (see Fig. 1.14).
From a signal processing perspective, this operation corresponds to the *time compression* and *time expansion* methods, which formalize how a sound waveform $x(t)$ can be shortened or lengthened without altering its perceptual identity [23].

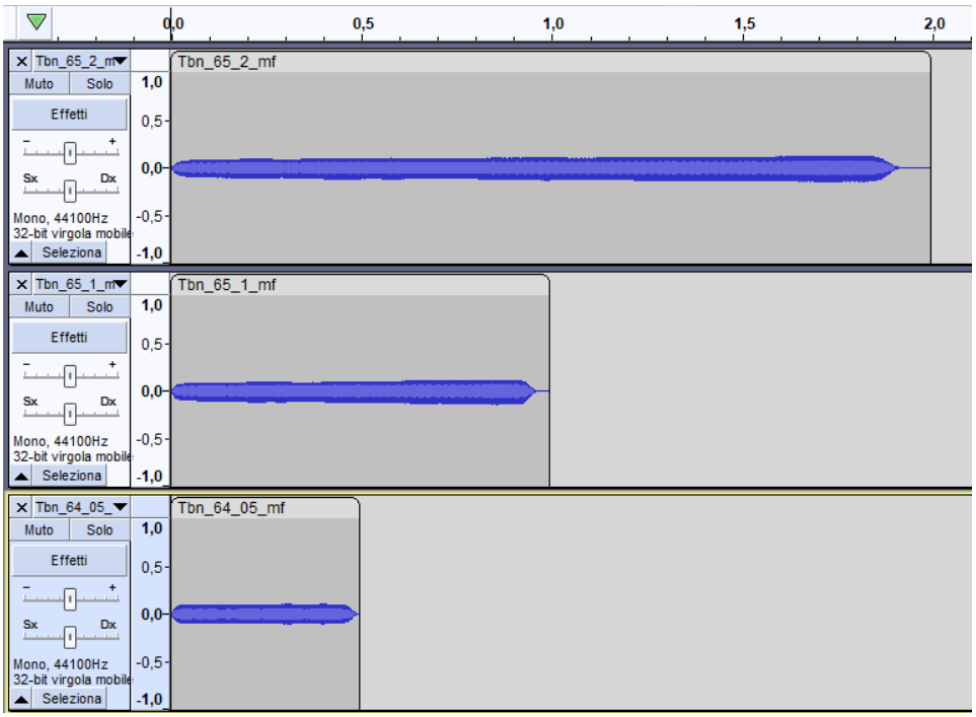


Figure 1.14: Example of the same note of a trombone with different durations.

Time compression and time expansion are achieved by modifying the playback sampling frequency $f_{s,\text{play}}$. Consider a signal originally sampled at $f_{s,\text{rec}}$; reproducing it at a different sampling rate produces a temporal scaling governed by

$$f_{s,\text{play}} = v\, f_{s,\text{rec}}. \tag{1.14}$$

As a consequence, the signal duration is inversely scaled, yielding a time–scaled version with duration

$$T_{\text{play}} = \frac{T_{\text{rec}}}{v}. \tag{1.15}$$

However, this operation inevitably modifies the spectral content of the signal: compression ($v > 1$) raises the pitch, whereas expansion ($v < 1$) lowers it. Since pitch preservation was essential for musical coherence in this project, these methods were not suitable.

For this reason, the system adopts the second class of techniques **time stretching**: the method allows modification of the temporal duration without affecting pitch, because the goal is to obtain a new signal $x'(n)$ from an original signal $x(n)$ such that

$$M' = \alpha M,$$

where M is the number of samples of the input sound and α is the desired scaling factor (e.g., 0.5 for half duration, 2 for double duration). In the script, this operation has been implemented using the time–stretching functions provided by the librosa library [4], which modify the temporal axis while preserving the harmonic structure of the sample.

Through this method, each original note (approximately one second long) was transformed into two additional versions with durations of 0.5 s and 2 s, allowing each instrument to be available in three rhythmic granularities while maintaining pitch accuracy, and avoiding artifacts such as formant shifting.

In the proposed framework each duration is equal to a different music rhythm and, matching seconds and pulses, matching velocities can be seen in table 1.6.

[4]Librosa is a python package for music and audio analysis. It provides the building blocks necessary to create music information retrieval systems.

2 | Results

This chapter presents graphical analyses of the results obtained from the swarm robotic orchestra. In particular, the behavior of the agents across the four asynchronous modules are examined in order to determine whether the robotic orchestra successfully achieved the intended objectives.

The simulator allows the user to run multiple simulations and generate the corresponding csv output files, or to produce a video of the resulting performance, which is automatically displayed at the end of the interaction. The large volume of generated data is necessary to analyze the simulators behavior across a wide range of experimental conditions and to assess the stability of the framework under different initial configurations.

To support the analysis of simulation results, a graph generation module has been implemented through the `analyzer.py` class. This component uses Python libraries such as matplotlib, pandas, and numpy to produce visualizations that facilitate the evaluation and interpretation of simulation outcomes.

The next section analyses the results obtained from a series of simulation runs, comparing different parameter configurations. Each configuration was simulated 25 times with varying durations to ensure a sufficiently robust dataset for evaluation.

To conclude, the results address the phase, beat, harmony, and timbre modules, providing a comprehensive overview of the systems behavior under varying configurations. In particular, the swarm robotic orchestra is expected to achieve coordination in rhythm, phase, harmony, and timbre distribution, and the time required to reach each form of synchronization is evaluated.

A further objective is to examine how the bio-inspired algorithms implemented in the beat and timbre modules integrate into the pre-existing system architecture: this analysis wants to verify the coexistence and mutual influence of musical structure, collective dynamics, and bio-inspired control principles.

2.1. Phase synchronization

Phase synchrony, denoted as $\Delta\Theta(t)$, is used to quantify the degree of phase alignment among all robots at time t, and is defined as the mean normalized absolute phase difference between the phase counters of all robot pairs. Two robots are considered to be in a condition of maximum asynchrony when their phases are opposite, for instance when robot i has phase counter $\theta_i(t) = 0$ and robot j has phase counter $\theta_j(t) = \theta_i(t) + \pi = \pi$. Therefore, the maximum possible phase difference between any two robots is π.

The phase synchrony measure $\Delta\Theta(t)$ is computed as

$$\Delta\Theta(t) = \frac{2}{M(M-1)} \sum_{i=1}^{M} \sum_{j=i+1}^{M} \left(|\theta_i(t) - \theta_j(t)| \bmod \pi \right), \tag{2.1}$$

where M denotes the total number of robots in the system. The normalization factor ensures that $\Delta\Theta(t)$ is bounded in the interval $[0, 1]$, where $\Delta\Theta(t) = 0$ indicates perfect phase synchronization among all robots, while $\Delta\Theta(t) = 1$ corresponds to maximum phase asynchronous.

In the previous implementation with a local distribution, interactions among individuals happened only once reached a certain threshold as distance parameter. During the first period of developing, this implementation has been replicated, and the most relevant parameters regarding interactions has been analyzed to study how swarm reaches phase synchronization (that was not divided in beats). Therefore, the variables evaluated to study phase synchronization were sensor parameter (to compute the necessary distance to share phase) and the swarm difference size.

Previous equation has been used to evaluate 3 different swarm sizes of 5, 10 and 15 robots with a fixed threshold value and 1 minute simulation duration. As can be seen from the plot in the Fig. 2.1, larger swarm robotic size ensures less time spent to reach synchronization, although system phase stabilization appears only after 40 seconds.

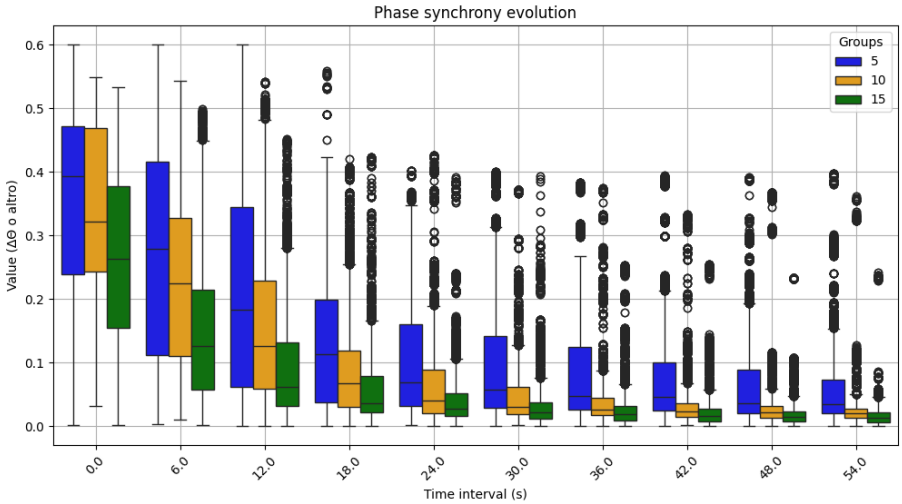


Figure 2.1: Phase synchronization evaluated for different swarm sizes.

For the threshold analysis, simulations were conducted using a group of 15 robots, with the threshold parameter set to 100, 200, and 400. As shown in Fig. 2.2, increasing this parameter leads to faster phase synchronization. This effect occurs because a larger threshold expands the robots effective communication range, allowing each agent to transmit its phase value to a greater number of neighbors.

Conversely, reducing the sensing range limits the number of robots that receive phase information, thereby increasing the time required to achieve phase synchronization.

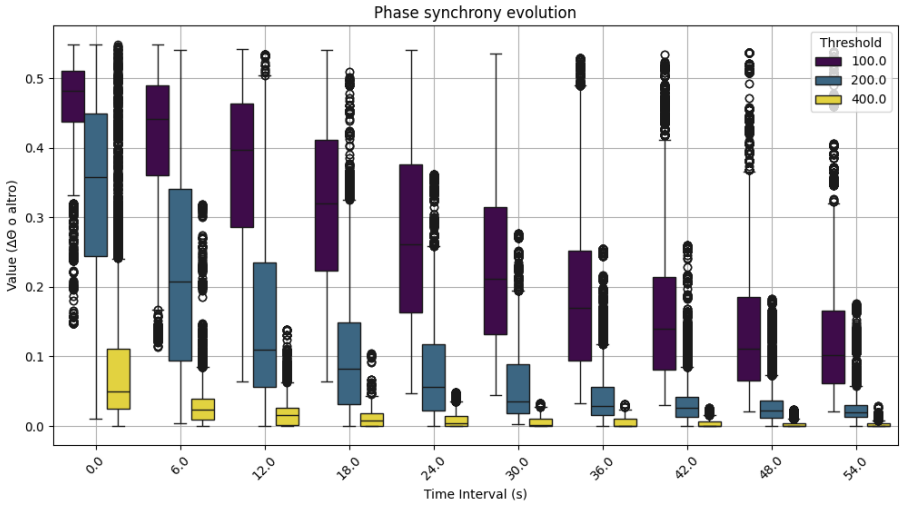


Figure 2.2: Threshold analysis for phase synchronization.

It can be deduced that, as the number of elements in the swarm or the distance required

to allow data exchange increases, the swarm reaches phase coordination more quickly; this is due to a greater availability of phase-related data in both cases.
In the current project interaction within robots is like global, therefore it is useless to study the value of the threshold.
The worst case for phase coordination is an experiment about 4 robots and 30 BPM: this means that communication occurs only each 2 seconds, and only among other 3 individuals. As can be noticed from the graph (see Fig. 2.4), increasing BPM value, synchronization is reached already after few seconds:

For what concerns the global communication, enabled by setting distance sensor parameter to an infinite value, phase synchronization analysis can be limited to the first ten seconds of the simulations. In particular, phase coordination has been evaluated for swarms of different sizes, specifically those consisting of 8, 16, and 32 robots. As can be seen from the graph (see Fig. 2.3), a larger group has smaller $\Delta\Theta(t)$ values, and consequently will achieve phase coordination even more quickly than a smaller group.

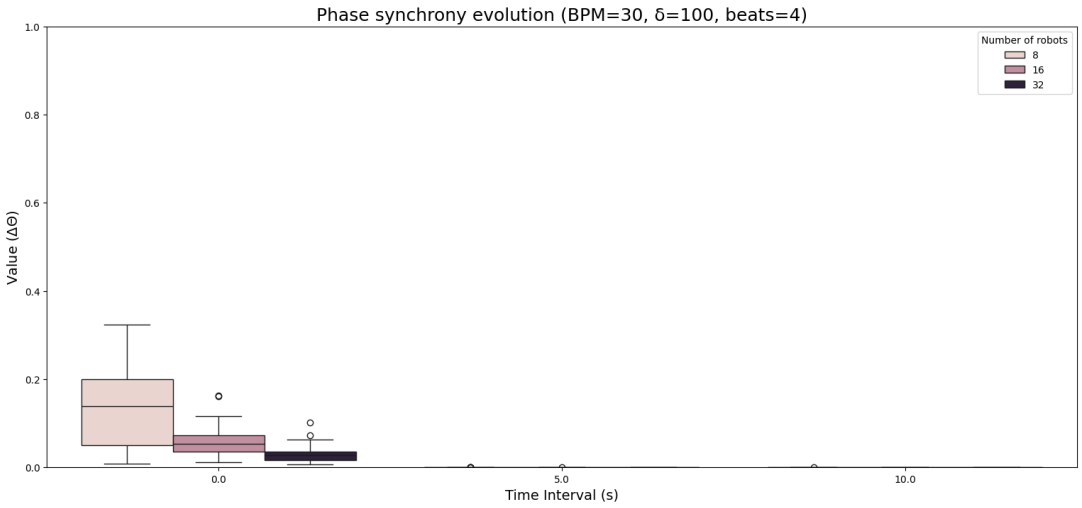


Figure 2.3: Phase synchronization among different group of robots.

Another parameter that significantly influences rhythmic coordination is the BPM, as it determines the temporal interval between successive notes. For instance, at 30 BPM a new beat occurs every 2 seconds, meaning that robots receive and store note related information at 2 second intervals; instead at 60 BPM, new information is processed every second, while at 120 BPM the interval is reduced to 0.5 seconds.
Consequently, under a 120 BPM configuration, phase synchronization is achieved more rapidly, as shown in Fig. 2.4, whereas under a 30 BPM configuration the convergence time is significantly longer.

This behavior is observed across all modules except the timbre module, where increased information redundancy does not necessarily reduce the time required to reach the target distribution.

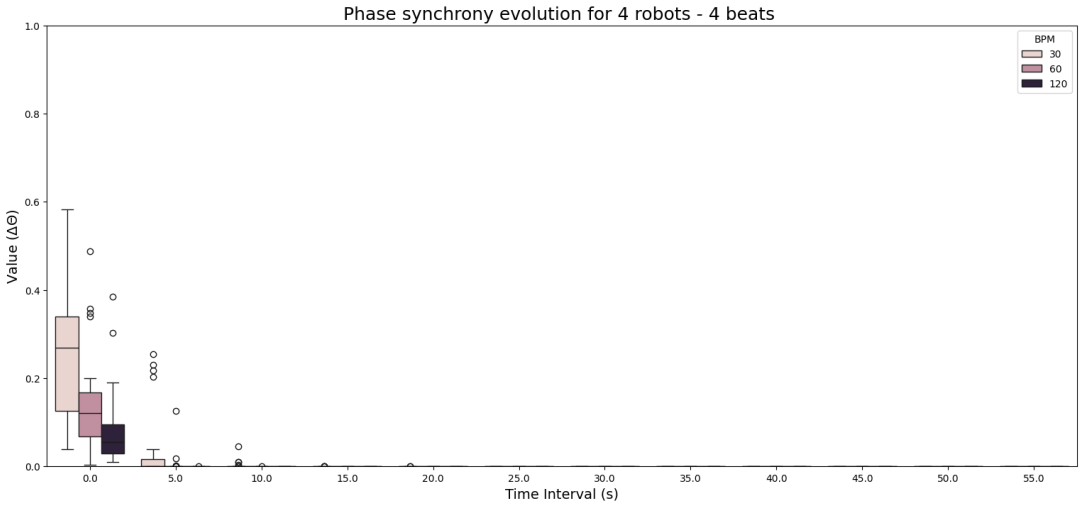


Figure 2.4: Phase synchronization among 4 robots for different BPM configurations.

Comparing the two communication strategies, global communication proves more effective than local communication, as the phase value is broadcast to all agents at each update rather than exchanged only among nearby neighbors.

2.2. Beat synchronization

To quantify beat synchronization at the swarm level, metric $\Delta B(t)$ is defined as the mean normalized absolute difference between the beat counters of all robot pairs: let $b_i(t) \in \{1, \dots, B_{\max}\}$ denote the beat counter of robot i at time t, where $B_{\max}$ is the maximum beat index; two robots are perfectly synchronized when $b_i(t) = b_j(t)$, while they are maximally asynchronous when one robot is at beat 1 and the other at beat $B_{\max}$, yielding a maximum difference of $B_{\max} - 1$. The swarm-level beat asynchrony is computed as:

$$\Delta B(t) = \frac{2}{M(M-1)} \cdot \frac{1}{B_{\max} - 1} \sum_{i=1}^{M} \sum_{j=i+1}^{M} \left| b_i(t) - b_j(t) \right|, \tag{2.2}$$

where M is the number of robots. The normalization ensures $\Delta B(t) \in [0, 1]$, with $\Delta B(t) = 0$ indicating perfect beat synchronization across the swarm, and $\Delta B(t) = 1$ indicating maximum beat asynchrony.

In music theory there is no intrinsic limit to the number of beats that a measure may contain; however, within simple time signature such as the one implemented in the proposed framework, introducing a number of beats that is a multiple of four is of limited practical relevance. For instance, defining eight beats corresponds to a temporal structure equivalent to two consecutive measures, which is more naturally represented as two separate bars rather than a single extended one. For this reason, the number of beats implemented in the framework is restricted to the range from 1 to 7.

To evaluate beat synchronization the parameters considered are swarm size and BPM, in order to ensure a consistent comparison with phase synchronization in the rhythmic component of the framework. Number of beats is irrelevant to examine beat module because, as the number of beats increases, the temporal resolution of the beat counter becomes longer, leading to a larger state space for synchronization and finally a $\Delta B(t)$ increase. Consequently, increasing the number of beats in the simulation results in a longer time required for the robots to achieve synchronization of their beat counters.

The first graph evaluates, as for the phase, how the number of individuals within a group influences the beat synchronization (with 8, 16 and 32 robots) with a configuration of 30 BPM (see Fig. 2.5). As for the previous module, increasing the number of robots speeds up the time it takes for all agents to count the same number of beats. It can be deduced that, after 15 seconds, every swarm robotic orchestra has reached beat synchronization.

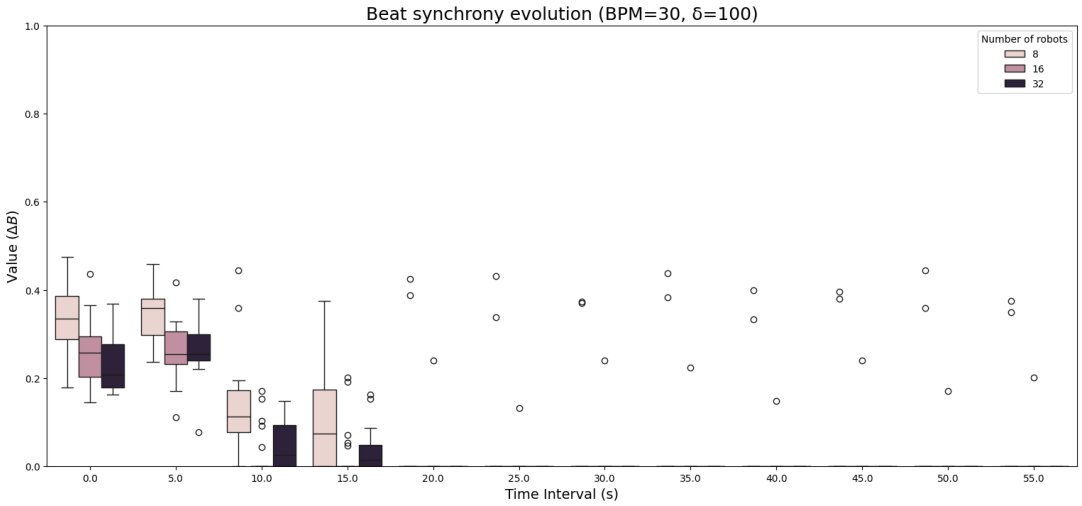


Figure 2.5: Comparison of beat synchronization for different swarm sizes with 30 BPM.

The second graph evaluates the same swarm size differences for a faster music velocity (60 BPM) (see Fig. 2.6), showing that the increase of available info reduces necessary amount of time to reach beat synchronization by 10 seconds.

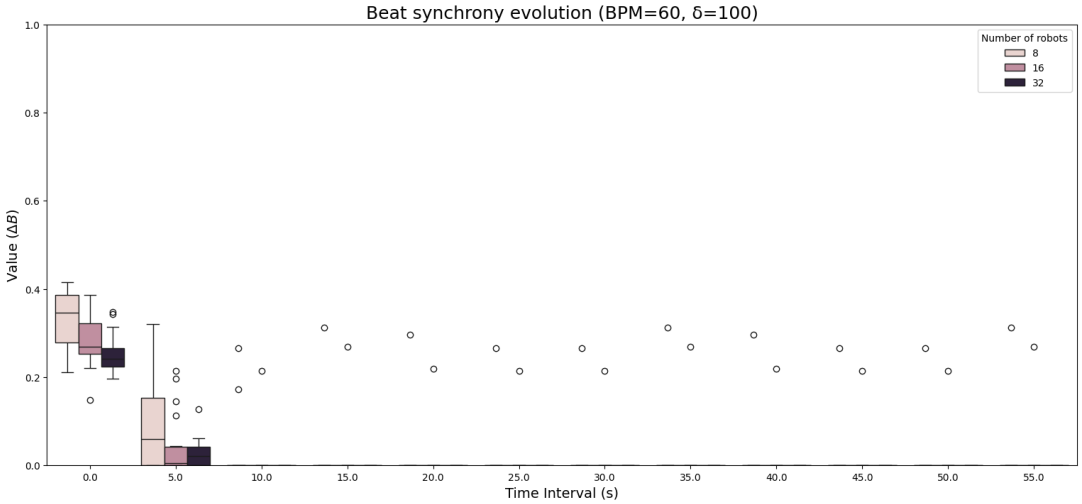


Figure 2.6: Comparison of beat synchronization for different swarm sizes with 60 BPM.

A variation of BPM from 30 to 120 is also analyzed for a group of 16 robots, that corresponds more or less to a plausible real experiment. As for the phase, 120 BPM corresponds to a large amount of data; therefore, is easier for robots coordinating their counter to the correct beat position (see Fig. 2.7).

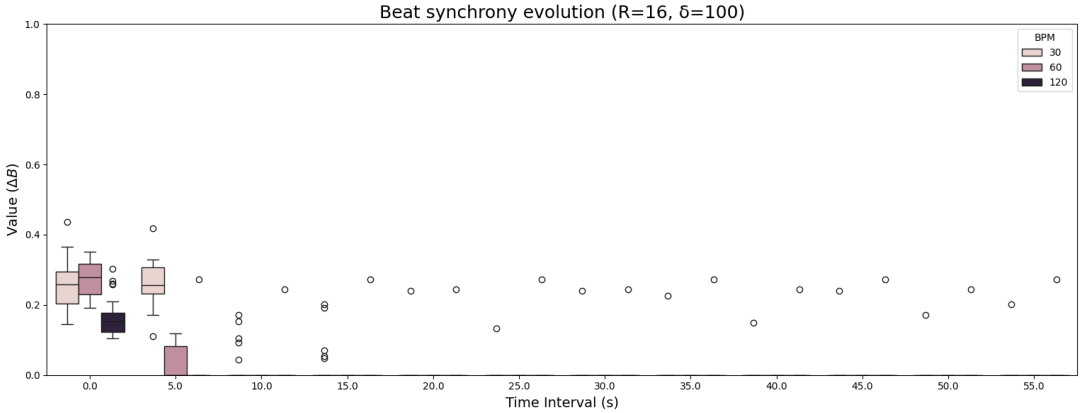


Figure 2.7: Beat synchronization with different BPMs.

From results, is deducible that swarm reaches beat synchronization during the first seconds of the simulation: worst results came from slowest velocity, on which notes follow one another every two seconds.

2.3. Harmony synchronization

To measure the capability of the robotic swarm to reach an harmonic synchronization over time, *harmonic agreement* metric $H(t)$ is introduced to observe this behavior over time, because the metric measures the degree to which the robots converge toward, at least, a common musical scale. The sequent parameters are defined for the metric: M denotes the number of robots in the swarm, and each robot $i \in \{1, \dots, M\}$ produces a musical pitch at a given time instant t. Pitch is represented by

$$p_i(t) \in \{0, 1, \dots, 11\},$$

corresponding to the twelve pitch classes of the chromatic scale.

Let $\mathcal{F}$ be a predefined family of musical scales (e.g., major, pentatonic, or whole-tone); each scale family $\mathcal{F}$ is defined as a collection of pitch-class sets indexed by a root note:

$$\mathcal{F} = \{S_r \subseteq \{0, \dots, 11\} \mid r \in \{0, \dots, 11\}\},$$

where S_r denotes the scale with root pitch class r.

For each candidate scale $S_r \in \mathcal{F}$, $N_r(t)$ defines the number of robots whose pitches belong to that scale at time t as

$$N_r(t) = \left|\{i \in \{1, \dots, M\} \mid p_i(t) \in S_r\}\right|.$$

The harmonic agreement at time t is then defined as the maximum proportion of robots whose pitches belong to at least one common scale within the family:

$$H(t) = \frac{1}{M} \max_{r \in \{0,\dots,11\}} N_r(t) \tag{2.3}$$

By construction, the metric satisfies

$$H(t) \in \left[\frac{1}{M}, 1\right] \tag{2.4}$$

The lower bound $H(t) = \frac{1}{M}$ corresponds to the case in which each robot plays a pitch belonging to a different scale, indicating complete harmonic disagreement. The upper bound $H(t) = 1$ indicates full harmonic consensus, where all robots produce pitches belonging to the same musical scale.

In the experimental evaluation, $H(t)$ is computed at discrete time intervals. Within each interval, the last pitch produced by each robot is considered, ensuring that the metric reflects the instantaneous collective harmonic state of the swarm.

As a first approach, harmonic agreement has been evaluated with swarms of different sizes with 30 BPM configuration, playing notes from major scales (see Fig. 2.8).

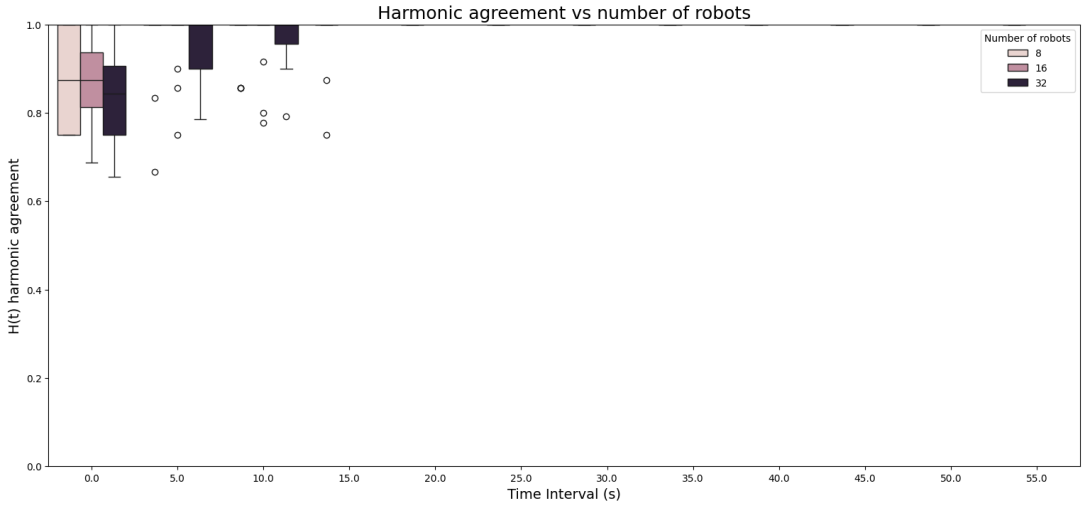


Figure 2.8: Harmonic agreement for different swarm sizes, with 30 BPM.

It can be noticed that the worst case corresponds to the largest swarm group of 32 robots, that reaches harmonic consensus within the first 10 seconds; for what concerns other configurations, results are even faster.

A second evaluation has been done for different scale types, to study If different patterns could affect or not results; additionally, simulations are developed with a 60 BPM configuration. In the following graph boxplot regarding the three implemented scales are evaluated (see Fig. 2.9):

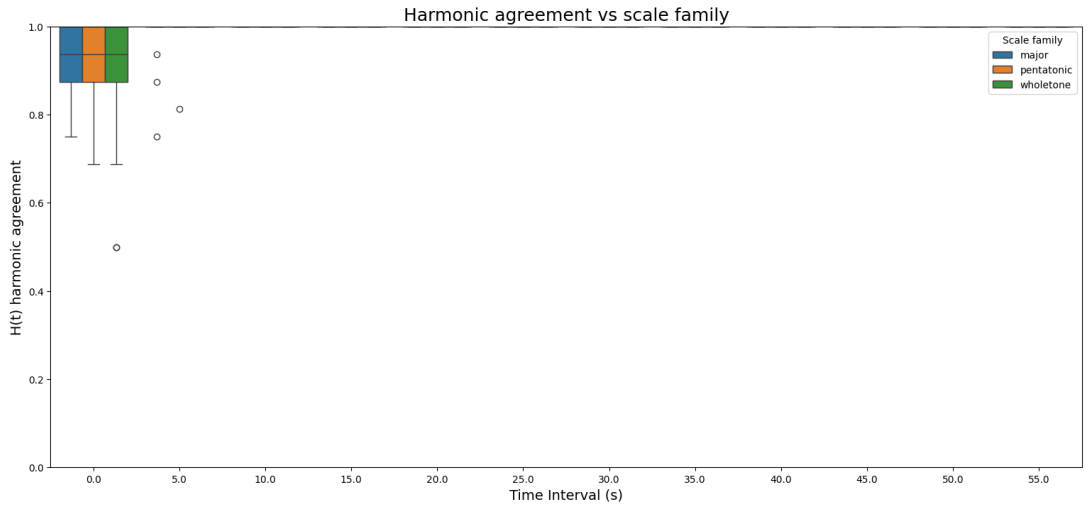


Figure 2.9: Harmonic agreement for different musical scales, with a swarm of 16 agents and 60 BPM.

It can be observed that changing the scale family does not produce significant variations in the $H(t)$ results, as the distributions remain largely comparable. Compared to the previous plot, the main difference concerns the temporal evolution of the harmonic agreement, which emerges within the first five seconds as a consequence of the increase in tempo from 30 to 60 BPM.

A different approach is to evaluate module agreement for different numbers of beats implemented in the framework, in particular 4, 5, 6 and 7 (see Fig. 2.10).

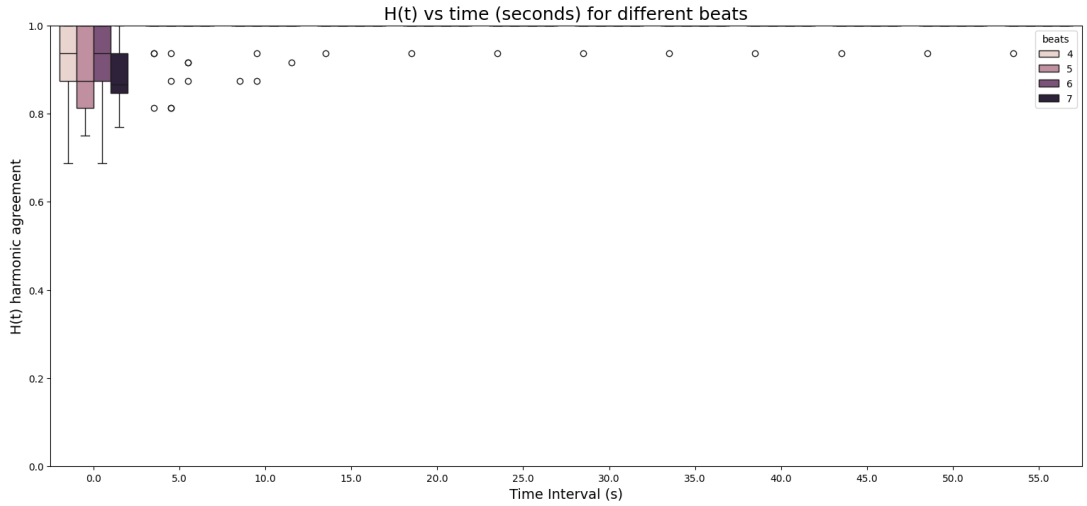


Figure 2.10: Harmonic agreement for different number of beats, with a swarm of 16 agents and 60 BPM.

The worst-case scenario corresponds to the configuration with 7 beats per bar, which, over a simulation lasting one minute, results in fewer than nine measures. Consequently, each robot is exposed to a smaller number of musical events compared to configurations with fewer beats per bar. Table below reports a comparison of the different beat configurations 2.1.

Beats per bar	Bars per minute	BPM
4	15	60
5	12	60
6	10	60
7	8.57	60

Table 2.1: Number of bars per minute for different beat configurations at 60 BPM

2.4. Timbre synchronization

Regarding last module described, boxplot graph has been implemented in order to analyze timbre distribution across robots, with an analysis frequency of 30 seconds. Remembering the equation based on instruments distribution 1.10 and the structure to reach the correct instruments distribution 1.5, simulation has been evaluated for duo, trio and quartet ensembles with a configuration of 3 minutes duration, $\frac{4}{4}$ as time signature and 60 BPM velocity. This choice is motivated by the experimental conditions achievable with real robots which, in the case of an emergent swarm robotic orchestra, are likely to involve approximately 16 units.

The most important parameter to examine is δ. As defined in Equation **??**, this parameter represents a variation of the response threshold reinforcement model, and is used to quantify the difference between the current swarm distribution and the target distribution.
The parameter δ regulates the stimuli update associated with each instrument and, as it is not present in the related literature, it has been analyzed under different configurations to identify potential mathematical relationships with other variables in the framework.

The first studied ensemble is the *duo*, made by violin and viola played by 8 robots: as can be seen from the plots, swarm reaches target distribution quite fast during the first minute and, increasing δ value, timbre synchronization performs in less time. In this experiments, 40, 60, 80 and 100 have been evaluated as maximum δ (see Fig. 2.11).

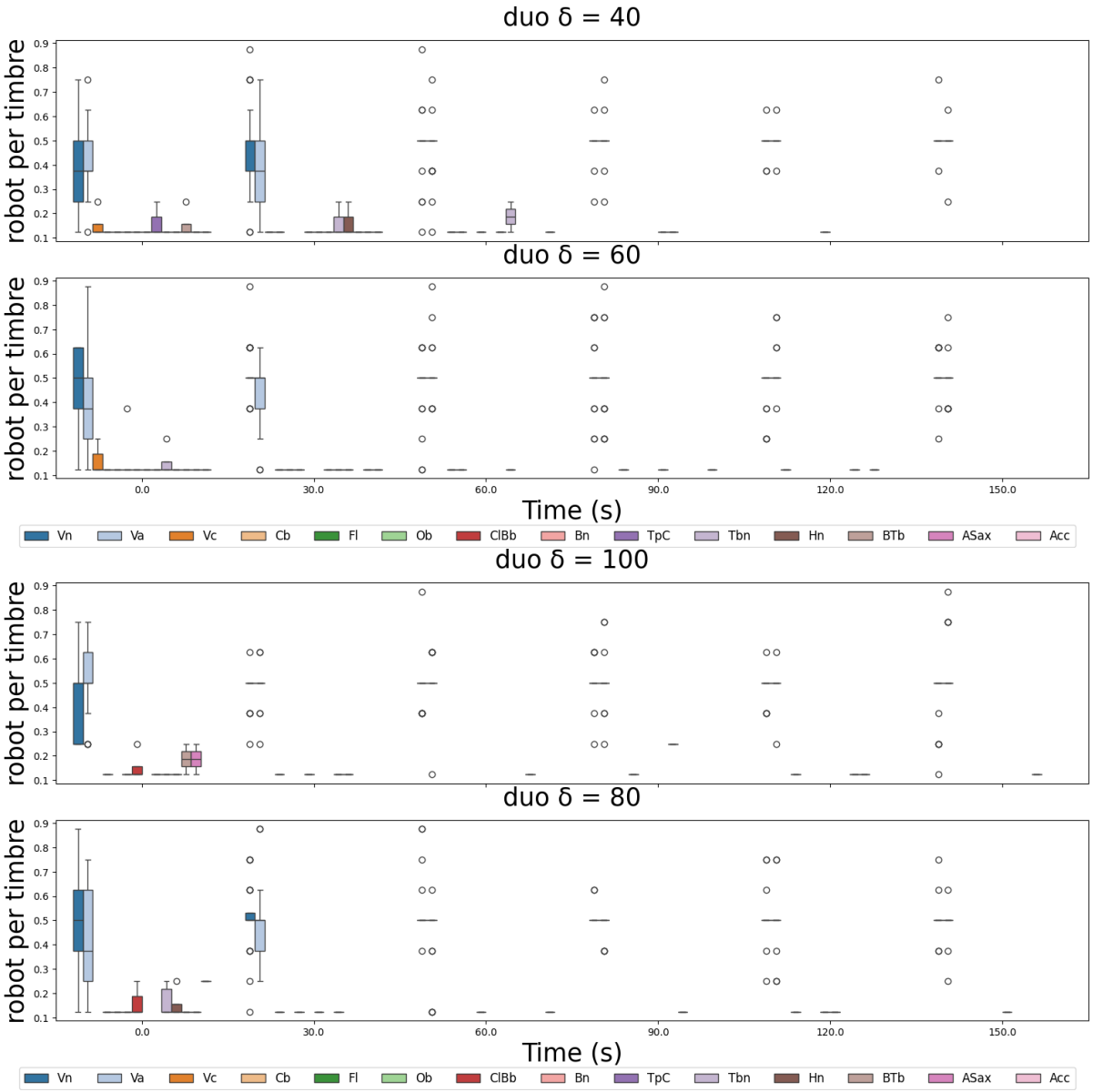


Figure 2.11: Comparison of duo configurations for different δ values.

As can be highlighted from plots, the best performance for the swarm is with $\delta = 100$: robots divide themselves in a balanced proportion between violin and viola players within the first 30 seconds.

Next configuration is about *trio*, made by flute, clarinet and violoncello performed by 12 robots (see Fig. 2.12).

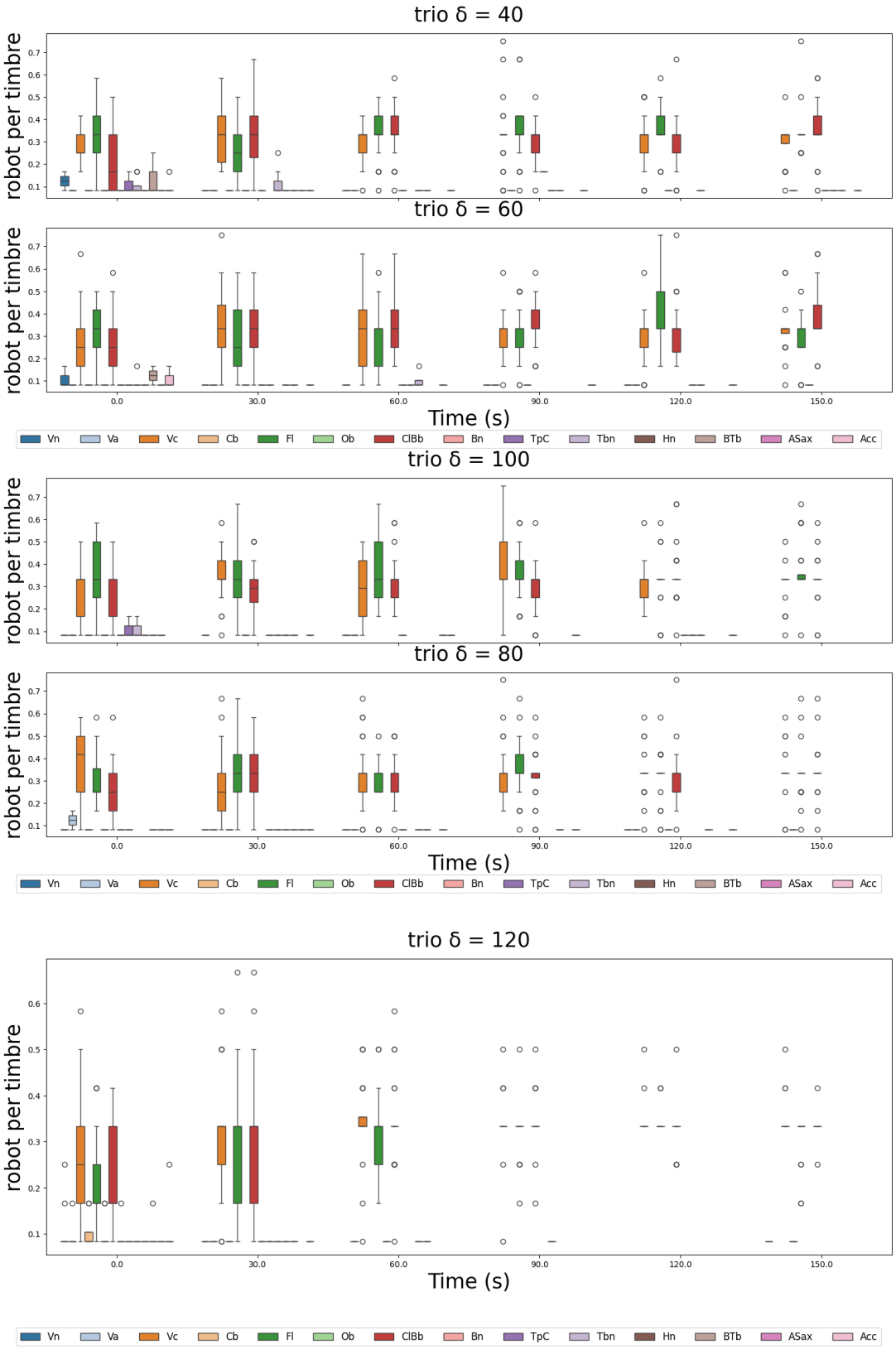


Figure 2.12: Comparison of trio configurations for different δ values.

For this configuration, results are less stable than before but, as for what observed in the previous case, increasing δ parameter corresponds to a better result, as can be seen in the last trio plot with $\delta = 120$.

For what concerns *quartet* ensemble, the precedent values analyzed for duo and trio did not work for a swarm with 16 agents; therefore, δ has been studied starting from 120 (see Fig. 2.13).

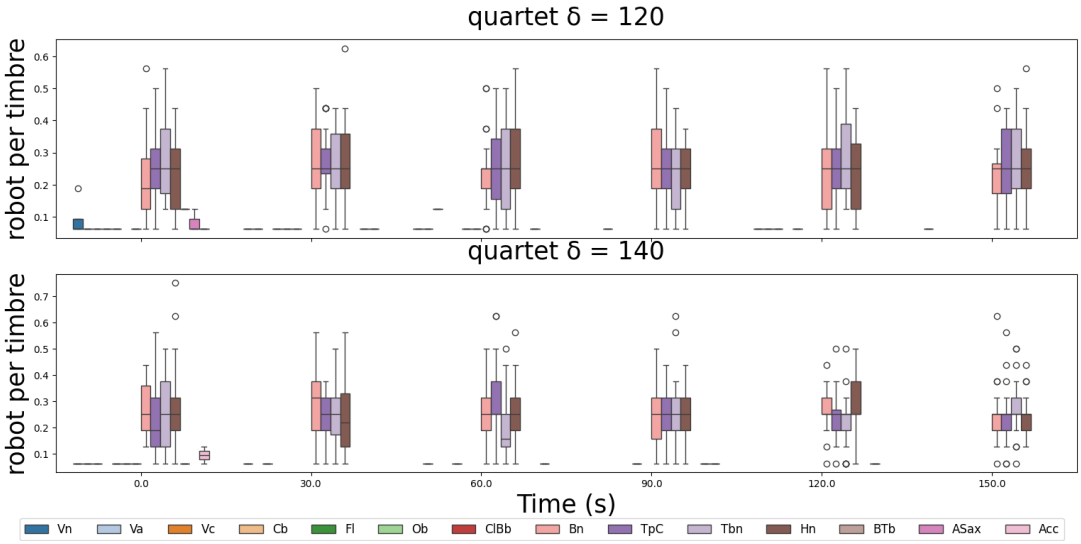


Figure 2.13: Comparison of quartet configurations for 120 and 140 δ values on 60 BPM.

Trying to enhance results, the same configuration with a 120 BPM velocity has been implemented: the speed music variation entails more communication between robots, who have a greater quantity of information available to them relating to the other agents. The simulation lasts 3 minutes with delta values as 140 and 160 (see Fig. 2.14).

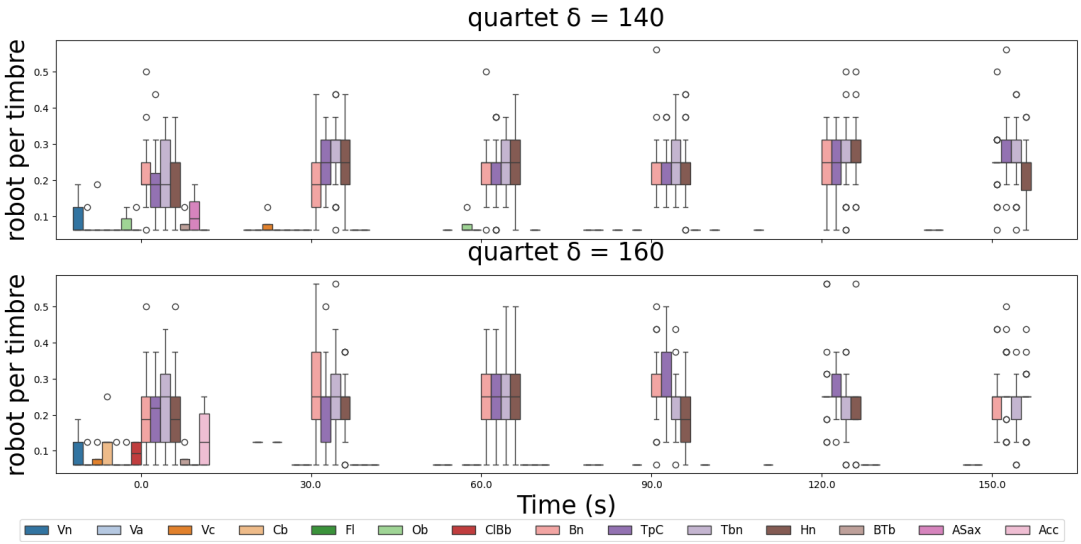


Figure 2.14: Comparison of quartet configurations for different δ values on 120 BPM.

It can be observed that, in musical ensembles with a small number of instruments, increasing the value of δ leads to faster convergence toward the target distribution, thereby facilitating timbre synchronization. This relationship, however, is not observed in ensembles with a larger number of instruments, such as a quartet.

A possible explanation lies in the amount of timbre-related information processed by each robot, particularly in simulations involving a larger number of agents. The stimuli update mechanism is triggered whenever a robot plays, affecting all other agents. If the swarm is not evenly balanced with respect to the implemented beat structure, the resulting redundancy of information may influence the instrument selection process and hinder stable timbre allocation.
To conclude, the random instrument selection described in Eq. 1.5 inevitably influences the results obtained.

3 | Discussion

This chapter outlines the final conclusions regarding the project, referring to the results reported in the previous chapter and the general structure of the work. Subsequently, some considerations regarding potential improvements applicable to the framework in future developments will be profiled.

3.1. Conclusion

The proposed framework emerges from the interaction of multiple disciplines, most notably music, swarm robotics and biology. As discussed in the previous chapter, algorithms inspired by collective animal behavior exhibit high stability within the proposed system, particularly at the fundamental layers of rhythm, beat and harmony. The swarm typically reaches coordination rapidly in all modules, especially with respect to beat, phase and harmonic synchronization, whereas timbre distribution requires a longer convergence time.

As outlined in the introductory sections, this thesis is based on the development of a framework designed to:

- simulate the behavior of the physical robots implemented within the Emergent Orchestra project;
- extend the musical capabilities of the original system.

The research activity carried out in this thesis has therefore been predominantly implementation oriented. The results obtained confirm the consistency of the proposed framework with the pre-existing project, since the framework provides stable and reliable behaviors for all the originally implemented modules. With regard to the experimental analysis, the newly introduced modules exhibit promising results, even if requiring longer convergence times. Furthermore, an additional feature has been developed to generate a multimedia output, which can be exploited both as a validation tool and as an artistic aim.

Overall, the framework ensures coherent and effective swarm synchronization across the four modules considered. Swarm robotic orchestra also highlights the complementarity of the disciplines involved in the project, the systems scalability across different configurations, and its modularity, which allows individual modules to be activated or deactivated through the configuration file. Overall, the proposed framework provides a solid foundation for future developments and extensions.

It is important to note that the current software implementation does not account for certain delays that would inevitably arise in a real-world deployment, such as latency in note transmission and perception across the swarm. In the present framework, each robot computes its phase based on the exact millisecond at which a note is generated, resulting in an unrealistically high level of hardware reactivity that would need to be addressed in future implementations.

A video of some simulations run with different configurations is available scanning this qr code.

Figure 3.1: qr code of a video about simulations run with different configurations.

3.2. Future developments

In this section are listed some plausible future additions that could enhance final musical result: these variation are strictly linked to the software implementation.

- **Add instruments to database**: The actual database has 13 instruments, mainly sourced from orchestra context (except for accordion and alto sax). This set can be fleshed out by adding samples of instruments used predominantly in other ensembles, such as guitar, synthesizers, or other types of saxophone. Percussion instruments would also be an important addition, as they would further expand the range of sounds the swarm can produce.
- **Implement the concept of pause**: From the moment the swarm achieves beat and phase synchronization, the performance generated by the robots lacks moments of silence, as each beat is continuously occupied by notes whose duration spans the

entire movement. This behavior is musically unrealistic, since musical structure is defined not only by notes but also by rests, which are essential for conveying directionality, articulation, and expressive motion within a performance.

- **Uncorrelating note duration from musical velocity**: Similarly to the harmonic dimension, the rhythmic structure of the simulation becomes predictable once coordination is achieved. To introduce variability at this level, robots could be allowed to play notes with different durations (metrics), rather than being constrained to a single note-length configuration associated with a given velocity signature. This decoupling would increase the complexity of the rhythmic layer of the framework, enabling richer temporal patterns and reducing structural predictability after synchronization.
- **Change of metric during simulation**: Related to the previous point, another approach to modifying the rhythmic structure of the simulation consists in allowing changes in time signature during the performance. This practice is common in most musical compositions structured into sections, where variations in meter are used to delineate themes and melodic ideas performed by the orchestra. Introducing dynamic time signatures would enhance structural articulation and contribute to a more realistic and expressive temporal organization of the robotic performance.
- **Implementation of new musical rules**: In the current project, robots assume the first beat as the primary reference for beat synchronization, in accordance with an implicit convention of Western tonal music. This assumption is inverted in other musical styles, such as jazz or blues, where rhythmic emphasis is often displaced or syncopated. Extending the robots knowledge base with additional musical rules would allow the swarm to dynamically alter the stylistic characteristics of the generated music during the simulation.
- **Implement a melodic module**: An important extension of the robotic behavior would be the introduction of a module dedicated to melodic line generation. This feature would build upon the inclusion of pauses and variable note durations discussed previously and would be activated once harmonic consensus has been achieved. Such a module would significantly increase the structural complexity of the generated music, reducing repetitive patterns that emerge as a consequence of harmonic synchronization and enabling the emergence of more articulated and expressive melodic developments.

With regard to the hardware aspect, a potential extension of the robotic platform would be the integration of inputs not only from homogeneous robotic devices but also from

human performers. This capability would increase the range of musicians with whom the robots could interact, enabling hybrid humanrobot ensembles and fostering interactive performances that actively involve the audience. Nevertheless, the incorporation of increasingly complex functionalities and musical knowledge into robotic agents must be accompanied by a corresponding assessment of the robots memory and computational constraints.

The code of the framework is available scanning this qr code.

Figure 3.2: qr code of GitHub link

Bibliography

[1] C. Arney. Sync: The emerging science of spontaneous order. *Mathematics and Computer Education*, 41(2):172, 2007.

[2] A. Attanasi, A. Cavagna, L. Del Castello, I. Giardina, T. S. Grigera, A. Jelić, S. Melillo, L. Parisi, O. Pohl, E. Shen, et al. Information transfer and behavioural inertia in starling flocks. *Nature physics*, 10(9):691–696, 2014.

[3] A. Baronchelli. The emergence of consensus: a primer. *Royal Society open science*, 5(2):172189, 2018.

[4] M. S. Barrett, A. Creech, and K. Zhukov. Creative collaboration and collaborative creativity: A systematic literature review. *Frontiers in Psychology*, 12:713445, 2021.

[5] T. Blackwell and M. Young. Self-organised music. *Organised sound*, 9(2):123–136, 2004.

[6] D. Bratsun and K. Kostarev. Thermal convection in huddling emperor penguins. *arXiv preprint arXiv:2508.16586*, 2025.

[7] M. Bretan and G. Weinberg. A survey of robotic musicianship. *Communications of the ACM*, 59(5):100–109, 2016.

[8] R. B. Dannenberg, H. B. Brown, and R. Lupish. Mcblare: a robotic bagpipe player. In *Musical robots and interactive multimodal systems*, pages 165–178. Springer, 2011.

[9] S. Feld. A generative theory of tonal music, 1984.

[10] D. J. Grout and C. V. Palisca. *A History of Western Music*. W. W. Norton & Company, 9th edition, 2014.

[11] H. Guo, T. Wu, and X. Xu. Sample-data output consensus for heterogeneous linear multi-agent systems with time-varying communication delays. In *2025 4th Conference on Fully Actuated System Theory and Applications (FASTA)*, pages 769–774. IEEE, 2025.

[12] G. Hoffman and G. Weinberg. Shimon: an interactive improvisational robotic

marimba player. In *CHI'10 Extended Abstracts on Human Factors in Computing Systems*, pages 3097–3102. 2010.

[13] P. Isley. The title of the work. How it was published, 7 1993. An optional note.

[14] Y. Kuramoto. Self-entrainment of a population of coupled non-linear oscillators. In *International symposium on mathematical problems in theoretical physics: January 23–29, 1975, kyoto university, kyoto/Japan*, pages 420–422. Springer, 2005.

[15] Y. Kusuda. Toyota's violin-playing robot. *Industrial Robot: An International Journal*, 35(6):504–506, 2008.

[16] H. Kuttruff. *Acoustics: an introduction*. CRC Press, 2007.

[17] M. Mauch, R. M. MacCallum, M. Levy, and A. M. Leroi. The evolution of popular music: Usa 1960–2010. *Royal Society open science*, 2(5):150081, 2015.

[18] F. Mondada, M. Bonani, F. Riedo, M. Briod, L. Pereyre, P. Rétornaz, and S. Magnenat. Bringing robotics to formal education: The thymio open-source hardware robot. *IEEE Robotics & Automation Magazine*, 24(1):77–85, 2017.

[19] N. Poltronieri. *Lezioni di teoria musicale*. Accord for music, 2002.

[20] A. Reina, E. Ferrante, and G. Valentini. Collective decision-making in living and artificial systems: editorial. *Swarm Intelligence*, 15(1-2):1–6, 2021.

[21] L. B. Rosselló, M. Alkilabi, E. Tuci, H. Bersini, and A. Reina. Emergent orchestras: A modular framework for musical robot swarms. In *Artificial Life Conference Proceedings 36*, volume 2024, page 29. MIT Press One Rogers Street, Cambridge, MA 02142-1209, USA journals-info , 2024.

[22] R. Sarfati, K. Joshi, O. Martin, J. C. Hayes, S. Iyer-Biswas, and O. Peleg. Emergent periodicity in the collective synchronous flashing of fireflies. *Elife*, 12:e78908, 2023.

[23] A. Sarti, L. Bianchi, and A. Bernardini. Sound modeling: Signal-based approach (part 1). sound analysis, synthesis and processing, Apr. 2023. Lecture slides, Module 1 – Sound Synthesis and Spatial Processing, M.Sc. in Music and Acoustic Engineering, Politecnico di Milano.

[24] R. K. Sawyer. *Group creativity: Music, theater, collaboration*. Psychology Press, 2014.

[25] J. Solis and K. Ng. Musical robots and interactive multimodal systems: An introduction. In *Musical Robots and Interactive Multimodal Systems*, pages 1–12. Springer, 2011.

[26] K. Tatar and P. Pasquier. Musical agents: A typology and state of the art towards musical metacreation. *Journal of New Music Research*, 48(1):56–105, 2019.

[27] R. Taw. Ants collective intelligence: What could we learn? *Science Insights*, 46(1): 1711–1722, 2025.

[28] G. Theraulaz, E. Bonabeau, and J.-N. Denuebourg. Response threshold reinforcements and division of labour in insect societies. *Proc. Biol. Sci.*, 265(1393):327–332, 1998.

[29] TinySOL. Instrumental sounds dataset, 2025. URL `https://forum.ircam.fr/projects/detail/tinysol/`. Accessed: 2025-04-09.

[30] V. Trianni, D. De Simone, A. Reina, and A. Baronchelli. Emergence of consensus in a multi-robot network: from abstract models to empirical validation. *IEEE Robotics and Automation Letters*, 1(1):348–353, 2016.

[31] L. Xie and X. Zhang. Dynamic leadership mechanism in homing pigeon flocks. *Biomimetics*, 9(2):88, 2024.

[32] M. Zanoni. Multi-agent systems. Lecture slides, Course: CREATIVE PROGRAMMING AND COMPUTING, 2024. Accessed: 2025-09-29.

[33] M. Zanoni. Agents and reactive agents. Lecture slides, Course: CREATIVE PROGRAMMING AND COMPUTING, 2024. Accessed: 2025-09-29.

[34] Z. Zheng, Y. Zhou, Y. Xiang, X. Lei, and X. Peng. Emergence of collective behaviors for the swarm robotic through visual attention-based selective interaction. *IEEE Robotics and Automation Letters*, 2024.

List of Figures

List of Tables

Acknowledgements

Giunti alla conclusione del lavoro di tesi e, più in generale del percorso di studi, ci tengo a porre i miei ringraziamenti alle persone che mi hanno permesso di vivere questo percorso duro ma gratificante.

Un doveroso ringrazimento, sincero prima che isituzionale, va al relatore Prof. Antonacci, il quale non solo mi ha dato l'opportunità di vivere l'esperienza di tesi all'estero con il programma Freemover, ma si è sempre reso disponibile ogni qualvolta abbia richiesto delucidazioni riguardo il lavoro svolto, dandomi consigli utili e guidandomi alla realizzazione del progetto.

Ringrazio il supervisore della tesi Andreagiovanni Reina, che mi ha accolto nel team di lavoro del "The Centre for the Advanced Study of Collective Behaviour" all'università di Konstanz. Sin dal primo giorno ho potuto partecipare alle iniziative del cluster, che mi hanno fatto vivere in prima persona l'esperienza della ricerca scientifica: ho potuto partecipare a conferenze, assistere a seminari scientifici settimanali e presentare un poster della mia tesi durante una conferenza interna al cluster, tutte attività avvincenti e formative di cui probabilmente non avrei potuto far parte in altre situazioni. Queste attività, oltre che stimolanti, mi hanno permesso di conoscere ricercatori e studenti in ambiti totalmente diversi dal mio: la sperimentazione della multidisciplinarità ha permesso di allargare i miei orizzonti e le mie conoscenze.

Un sentito grazie va sicuramente a Lluc, dottornado che mi ha seguito durante i 9 mesi di lavoro, il quale non si è mai risparmiato in consigli, videochiamate e aiuti concreti durante la tesi. La passione comune per la musica ha permesso di instaurare un rapporto cordiale prima che accademico.

Sempre in relazione ai ringraziamenti accademici, non posso non menzionare Marco Fele, dottornado in visita presso il cluster durante il mio soggiorno: grazie al tempo dedicatomi, ho potuto inserire nella tesi insegnamenti e tematiche ascoltate durante i seminari del cluster, incrementando la parte di swarm collective behaviour presente nel progetto di tesi. Ringrazio inoltre Suet Lee, postdoc all'università di Konstanz, la quale si è sempre interessata al progetto contribuendo con spunti interessanti.

Ringrazio infine Heiko, Paolo, Alberto, Pranav, Sindiso, Simay, Ulash, Andrè, e tutti i ragazzi del cluster, i ragazzi della Big Band dell'uni e tutti i ragazzi con cui ho condiviso la quotidinanità: a tutti grazie di cuore.

Una menzione a parte va infine ai miei due compagni di avventura Gabri e Iac, senza i quali quest'esperienza non sarebbe stata la stessa e sicuramente meno spensierata.