UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS DE TELECOMUNICACIÓN

MASTER IN
SIGNAL THEORY AND COMMUNICATIONS

MASTER THESIS

Design and Development of Continuous-Time Recurrent Neural Networks Evolution for Cooperative Distributed Multi-agent Systems

Rafael Sendra Arranz

2021

UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS DE TELECOMUNICACIÓN

MASTER IN
SIGNAL THEORY AND COMMUNICATIONS

MASTER THESIS

Design and Development of Continuous-Time Recurrent Neural Networks Evolution for Cooperative Distributed Multi-agent Systems

Author
Rafael Sendra Arranz

Tutor
Álvaro Gutiérrez Martín

2021

Abstract

Muti-agent systems are composed by multiple intelligent and distributed agents that interact in order to solve problems that would be utterly challenging individually. A type of multi-agent system that uses simple and locally interacting robots is swarm robotics, a class of collective robotics inspired by societies of insects. Swarm robotics is a field of research of constant growth and interest that combines concepts from artificial intelligence and robotics. It studies the use of many simple distributed robots that collectively cooperate in order to solve complex tasks. Moreover, the design of robust yet simple communication mechanisms, that allow the cooperation through direct interaction among robots, is an important aspect of swarm robotics systems. This Master Thesis explores the design and implementation of a minimal communication system, composed by a message and its underlying environmental context. To assess the performance and versatility of the communication, four benchmark swarm robotics tasks, that require communication at some extent, are designed and solved. The robot controllers, defining the behavior of agents, are based on Continuos-Time Recurrent Neural Networks (CTRNN) evolved using evolutionary computation algorithms. In particular, Genetic Algorithm (GA) and Separable Natural Evolution Strategies (SNES) are used and compared. All the experiments are carried out using a simulated robotics software designed and implemented within the frame of this Master Thesis. An important objective of this Master Thesis is the analysis of the communication that emerges as a result of the evolution process, in each experiment. It is shown that the swarm, whose robots are controlled by the evolved neural controllers, is capable of successfully solving the tasks of all the experiments. SNES outperforms GA in three of the four designed tasks, based on the obtained mean fitness scores. Apart from a detailed analysis of the emerged behaviors and communications, the scalability and robustness of the solutions are assessed in each experiment. The imposed tests expose that the evolved neural controllers fulfill both scalability and robustness properties, which are highly desired features of swarm robotics systems. Besides, the communication mechanics resulting from evolution are remarkably diverse. Specifically, it is shown that depending on the task, the communication can be purely situated, abstract or a combination of both.

Keywords— Swarm Robotics, Evolutionary Computation, Continuous-Time Recurrent Neural Networks, Recurrent Neural Networks, Minimal Communication, Emerged Communication, Genetic Algorithm, Natural Evolution Strategy, Neural Controller, Neuroevolution.

Resumen

Los sistemas multi-agente están compuestos por múltiples agentes inteligentes y distribuidos que interactúan con el fin de resolver problemas que serían notablemente complejos de abordar individualmente. Una clase de sistemas multi-agente que usa robots simples e interacciones locales es la robótica de enjambre, que es un tipo de robótica colectiva inspirada por las sociedades de insectos en la naturaleza. La robótica de enjambre es un campo de investigación en constante crecimiento e interés, que combina conceptos de la inteligencia artificial y de la robótica. En concreto, estudia el empleo de múltiples robots distribuidos con interacciones locales que cooperan colectivamente para resolver tareas complejas. Además, el diseño de mecanismos de comunicación que sean simples y robustos al mismo tiempo es un aspecto importante en la robótica colectiva. Este Trabajo Fin de Máster explora el diseño e implementación de un sistema de comunicación minimalista, compuesto tanto por el mensaje como por su contexto en el entorno. Con el fin de verificar el rendimiento y la versatilidad de la comunicación, cuatro tareas de robótica de enjambre que requieren comunicación entre robots han sido diseñadas y resueltas. El controlador robótico, responsable de definir el comportamiento de los agentes, está basado en redes neuronales recurrentes en tiempo continuo evolucionadas usando algoritmos de computación evolutiva. En particular, algoritmos genéticos (GA), y estrategias de evolución natural separables (SNES) son usadas y comparadas. Todos los experimentos han sido realizados usando un simulador robótico, diseñado e implementado a lo largo de este Trabajo de Fin de Máster. Un objetivo importante es el análisis de la comunicación que, en cada experimento, emerge como resultado del proceso de evolución. Los resultados muestran que el enjambre de robots es capaz de resolver correctamente todas las tareas propuestas. Tomando como referencia el valor de fitness promedio, SNES obtiene mejores resultados que GA en tres de los cuatro experimentos propuestos. Además de un análisis detallado del comportamiento y comunicación emergentes, la escalabilidad y robustez de las soluciones son evaluadas en cada experimento. Las pruebas impuestas muestran que los controladores neuronales evolucionados cumplen satisfactoriamente las propiedades de escalabilidad y robustez, que son características altamente deseadas en sistemas de robótica enjambre. Por otra parte, los mecanismos de comunicación resultantes de la evolución son notablemente diversos. En concreto, la comunicación puede ser situada, abstracta o una combinación de ambas, dependiendo de la tarea a resolver.

Palabras Clave— Robótica de Enjambre, Computación Evolutiva, Redes Neuronales Recurrentes en Tiempo Continuo, Redes Neuronales Recurrentes, Comunicación Minimalista, Comunicación Emergente, Algoritmos Genéticos, Estrategia de Evolución Natural, Controlador Neuronal, Neuroevolución.

Agradecimientos

En primer lugar me gustaría agradecer a mi tutor, Dr. Álvaro Gutiérrez, por su dedicación y asesoramiento a lo largo de estos meses de duro trabajo. Álvaro, gracias por confiar en mí y apoyarme en todo momento y en todos los aspectos de este trabajo. También quiero mostrar mi agradecimiento hacia mi familia, en concreto mis padres, mi hermano y mi hermana, por apoyarme en todas mis decisiones y metas. Ellos han sido también fundamentales para el desarrollo y finalización de este Trabajo Fin de Máster .

Contents

List of Figures

List of Tables

List of Algorithms

List of Acronyms

ANN: Artificial Neural Network

CTRNN: Continuous-Time Recurrent Neural Network

ER: Evolutionary Robotics

EA: Evolutionary Algorithm

FSM: Finite State Machine

GA: Genetic Algorithm

ICC: Instantaneous Center of Curvature

IR: Infrared

JSON: JavaScript Object Notation

NES: Natural Evolution Strategy

NEAT: Neuroevolution of Augmenting Topologies

NGA: Natural Gradient Ascent

PDF: Probability Density Function

RNN: Recurrent Neural Network

RF: Radio Frequency

SR: Swarm Robotics

STDP: Spike-Timing-Dependent Plasticity

SNES: Separable Natural Evolution Strategy

TPR: True Positive Rate

TNR: True Negative Rate

WOSP: Wave Oriented Swarm Paradigm

xNES: Exponential Natural Evolution Strategy

Chapter 1

Introduction

This Master Thesis is framed within the intersection of the soft computing fields of swarm intelligence, evolutionary computation and artificial neural networks. More specifically, a set of cooperative tasks are proposed and solved using a swarm of distributed, self-organized robots whose artificial neural network driven behavior is optimized using evolutionary computation. Apart from the task resolution, we focus on the emergence and post-analysis of the communication interactions within the swarm that arise from a minimalist, constrained and situated communication system. This chapter starts with a detailed description of swarm robotics basics and state of the art relevant to this work. Subsequently, the different objectives and contributions that underline this Master Thesis are clearly stated. To conclude the chapter, we expose in a summarized way the content and main topics of the remaining chapters of the document.

1.1 State of the art

Swarm robotics (SR) [1] is the research field that, combining aspects of artificial intelligence and robotics, studies the use of many simple distributed robots that collectively cooperate in order to solve complex tasks. The tasks addressed by swarm robotics can be either intrinsically collective problems, that can uniquely be solved through interaction, or single agent problems whose results can be boosted in multi-agent setups. SR is widely inspired by how biological swarms work in nature [2], exposing the emergence of complex collective behaviors from local interactions. Some examples remarkably recurrent in swarm robotics are ant colonies, bee colonies, flocks of birds, fish schools or slime molds, among others. In [3], the authors defined a set of conditions that a robotics system must fulfill in order to be considered a swarm robotics system:

- *Autonomy of robots*: the system is decentralized and robots can freely interact with the environment
- *Large swarm sizes*: the swarms are composed by considerable amounts of agents.
- *Homogeneity*: the behavior, equipment and resources are the same for all the robots in the swarm.
- *Inefficiency and simplicity of single agents*: robots are simple and cannot solve tasks individually (or at least not efficiently).
- *Local communication and sensing of the environment*: robots in the swarm can only sense the environment or communicate with other agents within the local surroundings.

Moreover, apart from efficiency in solving the task, scalability, robustness and flexibility are highly desirable properties in SR systems (see [3]). These features should be taken

into consideration in the design and validation stages of SR. A multi agent system is scalable whenever its resulting performance barely decreases with the swarm size growth. Additionally, robustness measures the system capabilities to react to unexpected and undesired perturbations in the agents or in the environment. Finally, flexibility reflects the ability of agents to react with different behaviors or roles under different conditions.

The most important and widely explored problems in swarm robotics are subsequently listed and briefly explained. We refer to [4, 5, 6] for remarkably complete and detailed reviews on the most relevant swarm robotics tasks. Some of them are summarized hereafter. In aggregation task, all the robots in the swarm must group together so that swarm area is compacted to small region of the environment where robots are located (see [7, 8, 9, 10]). Additionally, flocking imitates bird flocks in nature so that robots navigate across the environment in a coordinated and compact way, with matching motion orientation and velocity. One of the most important contributions in the flocking task is the model proposed by Reynolds in [11], which is composed by the three simple rules of cohesion, separation and alignment. Flocking has been addressed in numerous works [12, 13, 14, 15], analyzing different algorithms, robot interactions or minimal required information. Foraging task (see [16, 17, 18, 19, 20, 21, 22] for several examples) poses a complex problem in which robots have to transport objects from food areas, whose positions are initially unknown, to a central nest area. Agents have to cooperate in order to solve the task proficiently. Role allocation is a problem devoted to dynamically distribute different subtasks or agent roles wherein the swarm members, spatially [23, 24, 25] or temporally [26, 27, 28]. Other highly relevant swarm robotics problems are shape formation [29], leader election [30, 31, 32] or collective object transport [33].

In swarm robotics, each robot is capable of locally sensing its environment and interacting with it and with other agents through actions. The mapping between sensed stimuli to actions is normally denoted as robot or agent controller. The controller, that defines the behavior or policy of robots, is one of the principal design parts in swarm robotics. Controllers are typically divided into two main types, namely, behavior-based and automatic design methods (see [5]). Behavior-based design methods are the traditional and most used design procedures in swarm robotics, in which the controller functioning is handcrafted in response to the application requirements. Finite State Machines (FSM), either deterministic or probabilistic, are behavior-based design methods that define a set of behavioral states of the robots and the stimuli dependent transition conditions among states. Probabilistic FSMs have been successfully employed to describe the functioning of controllers for foraging [19], aggregation [8, 34] or task allocation [21]. Additionally, it is equally relevant the use of virtual or artificial physics in the tasks involving robot motion. In artificial physics, the overall contribution of a set of virtual forces determines the direction of motion of agents. Artificial physics (see [35]) have been applied to solve, for instance, flocking [14, 15] and aggregation [36]. Alternatively, as the name denotes, automatic design methods use optimization algorithms in order to tune the parameters of some mathematical models describing the controller mapping. The optimization is usually driven by a fitness function or reward signal (depending on the family of algorithms), defined by the researcher, that specifies the goodness and suitability of the robot actions. Moreover, in the context of swarm robotics, the fitness is associated to the team or group of robots and not to each individual agent. The most important, and the procedure used in this work, is the use of evolutionary computation tools to optimize controllers in swarm robotics, commonly referred as Evolutionary Robotics (ER), see [37]. Moreover, automatic controller design in general, and ER in particular, are frequently coupled to Artificial Neural Networks (ANN), used as mathematical models to describe controller functioning. The optimization of ANN parameters by means of evolutionary computation is called neuroevolution, see e.g. [38]. Similarly, the resulting robot controllers

can be denoted as neurocontrollers or neural controllers. Some examples of simple neural controllers evolved using evolutionary algorithms, in diverse swarm robotics tasks, are the following [7, 39, 40]. Unlike in these studies, that address the evolution of feed-forward ANNs, the use of recurrent neural networks (RNN) is of special interest in swarm robotics. The main reason is that they allow action generation not only based on current stimuli being measured but also based on past experience and events. Specifically, Continuous-Time Recurrent Neural Networks (CTRNN), see e.g. [41], have been employed as neuro-controllers in multiple works [12, 42, 43, 44]. CTRNNs run in continuos time, making them remarkably suitable for swarm robotics, whose principal tasks intrinsically operate in continuos time. In [12], a CTRNN is evolved using a generational EA for the flocking of a swarm. They designed a fitness function that reflects cohesion, separation and alignment of the swarm, as in Reynolds' rules. Tuci et al [42] also optimize CTRNN controllers using an EA. In this case, they address a foraging problem, with a single nest and a single food area, in which robots have to decide whether they assume the role of foraging or nest patrolling. Moreover the task that they define imposes role switching within simulation to be completed correctly. Gutiérrez et al, proposed in [43] the evolution of CTRNNs using a generational GA for the task of heading alignment of robots. The study is firstly analyzed in simulated environments and validated with real robots. In [44] the authors use a simple EA with roulette wheel selection and a CTRNN controller in order to solve the problem of cooperative transportation of heavy objects. The works listed above, in the context of evolutionary computation tools applied to the optimization of ANNs, are either Genetic Algorithms (GA) or Evolutionary Algorithms (EA) [45]. Other algorithms from evolutionary computation have been also employed in swarm robotics and collecive robotics, such as differential evolution [46] or neuroevolution of augmenting topologies (NEAT) [47, 48]. Besides, a family of algorithms that has gained research interest in the context of single agent controllers are Natural Evolution Strategies (NES) (see [49]). In NES, the population individuals are randomly sampled from a search distribution whose parameters are optimized in order to maximize the expected fitness (see Chapter 2), following the natural gradient direction. However, up to our knowledge, NES algorithms have not been applied to the field of swarm robotics yet. In this Master Thesis we explore the use of both GA and Separable Natural Evolution Strategies (SNES), see [50], in order to evolve CTRNN neural controllers for different tasks.

Another critical design step in swarm robotics is the communication mechanics of the group. Communication within the swarm refers to any kind of interaction among robots in which information about states, actions or intentions of agents is shared across the swarm. According to [51], inter-agent communication in swarm robotics can be split into the following types:

- *Stigmergy*: or communication via the environment is, as its name suggests, an indirect type of communication in which the environment is used as the communication medium. In most cases, robots deposit traces, in the environment so that other agents can, eventually, acquire knowledge about their existence. A recurrent example of stigmergy in nature is the use of pheromones in ants to mark chemical trails to food. In swarm robotics, stigmergy has been frequently applied to foraging in the form of virtual pheromones (see e.g. [18, 20]).

- *Interaction via state*: is a type of communication in which each agent senses its neighboring robots throughout sensors. As in stigmergy, there is no explicit communication and the interaction is mainly due to existence awareness of other swarm members in the local surroundings. A typical example of interaction by means of sensing is the use of IR sensors acting as distance sensors (the sensor emits an IR ray and captures its reflection). In this situation, an agent can know the position of other robots relative to its own location (see [12, 13, 44, 47])

- *Direct communication*: is the inter-agent interaction by means of explicitly sending messages or information as in wireless networks. The messages can be directed to a particular robot or broadcasted. Several examples of direct communication in SR will be provided later. Direct communication can be combined with an interaction via the state. For instance, the message content can be accompanied by the message context, formed by, say, the direction of reception and the signal strength.

Additionally, a different division can be provided if the communication technology is considered. For minimal and short range communications, infrared (IR) technology is commonly employed (see [52]). IR sensors and emitters can be used for both distance estimation to solid objects [44, 47, 53, 54], direct communication [55] or both [43]. Apart from its short coverage, IR technology presents several issues such as interferences, ambient light distortion, communication death zones, impossibility to send and receive at the same time or low data rates (see [53, 56] for further details). However, it equally provides numerous advantages that make IR technology highly suitable for local communication in swarms of robots. Firstly, it is highly inexpensive and extremely low consuming, which, provided that swarm sizes can be arbitrarily large and individual robots are notably simple, are utterly desirable features in swarm robotics. Moreover, it allows a directionality awareness in the communication, as in-board IR communication in mobile robots is equipped with a set of receiver and emitters surrounding the robot perimeter. The knowledge of the direction from where neighboring robots are interacting with the agent is highly desirable in most applications and, more importantly, indispensable in many cases. Generally speaking, IR communication fulfills all previously exposed requirements for swarm robotics. Alternatively, there are other communication technologies that have been used in swarm robotics and multi-agent robotics. RF communication, encompassing a broad spectrum of technologies and protocols, is also a suitable communication mean when direction awareness is not an issue and long range global communication is required (see [57, 58, 59]). Finally, as a middle range communication alternative, several studies have addressed communication by means of sound (e.g. [60]) using speakers and microphones to produce and capture sound beeps. In the two lattermost kinds of communication, several of the exposed studies break the principle of locality presented at the beginning of the section. Although in a lot of applications long range coverage is utterly desirable, in many others it adds considerable communication overhead.

Another distinction can be highlighted in the case of direct communication. Essentially, we can differentiate between direct communication semantics and codes that are handcrafted by the researcher or communication semantics that arise from automatic controller design methods (e.g. ER). In the latter scenario, it is said that communication emerges. Clearly, the researcher still has to design and stablish the communication means and resources, albeit the semantics and the information relevant to the emerged communication is a result from optimization processes. Within this context, the taxonomy previously described according to [52] can be reformulated as follows (see [61]):

- *Abstract communication*: is a type of communication in which only the message content carries information. The environmental context associated to the message is either not processed or not relevant in the emerged communication. See e.g. [31, 62] and some of the experiments of [32].

- *Situated communication*: refers to communication scenarios in which both the message content and its corresponding environmental context carry information within the communication. Environmental context can be, for instance, the signal strength from which the distance can be estimated or the direction from where the message was received. Some examples of situated communication are [43, 63].

1.2 Objectives and contributions

In this Master Thesis, we study the type of communication (either abstract or situated) that emerges from evolution in a simulated environment of mobile robots. With this aim, we propose a series of benchmark problems that, at first sight, require some sort of interaction to be completed. The selected swarm robotics experiments are leader election, swarm frontier identification, orientation consensus and light follower. A review of the related work will be presented individually for each task in Chapter 5. The communication simulation resembles IR technology, and it has a number of simulated sensors uniformly distributed along the robot perimeter. We mentioned that the resulting communication can be either abstract or situated because in addition to the message content, which is a quantized real vector as it will be explained in subsequent chapters, the context underlying the message is also provided. The context varies depending on the task to be solved (among the proposed pool of experiments), but normally contains the signal strength, the direction of message reception and a communication state. The communication state will be defined in detail in Section 3.3, but, essentially, it establishes whether a robot sends its own message or relays the received message. The robot policy is controlled by a CTRNN evolved using either GA or SNES (both are analyzed) so that the messages are elaborated by the optimized CTRNN and the communication (situated or abstract) emerges as a consequence. Apart from solving the tasks, which is a secondary objective, one of our main aims is to assess the emerged communication in each experiment under minimal communication techniques. The communication is minimal mainly because of the following aspects:

- The communication is local, with a remarkably small communication range.
- Only one message can be received at each time instant. This means that regardless of the number of neighbors, the robot and, thus, the CTRNN is only aware of one neighbor at each simulation step.
- The possible directions of message reception or the number of IR sectors are restricted to 4 sectors, highly complexifying the tasks.
- The received context corresponds uniquely to the current received message.

All experimental setups of the tasks fulfill the requirements to be considered SR (see previous section). Moreover, in addition to an analysis of the evolved behaviors and the emerged communication, an scalability and robustness assessments are imposed to each solution. The main contributions are the following:

- A minimal SR system, with an utterly simple local communication is proposed. Its performance is assessed in several benchmarking tasks.
- GA and SNES algorithms are used to evolve the parameters of neural controllers. Although GA has been widely used in SR, SNES algorithms has not been explored, as far as we are concerned.
- The communication that emerges as a result of evolution is analyzed. Thereafter, the analysis reveals that sufficiently robust communication mechanics can emerge using such a simple communication system.
- The system versatility, scalability and robustness are tested, resulting in an SR framework that can be used in many collective robotics tasks.
- The resolution of the tasks is also an important contribution due to the robust and scalable behaviors that emerge using highly simple robots.

Another goal of this Master Thesis is the design and development of a robotics software simulator that encompasses the features and functions required by the presented swarm robotics experiments. The simulator, developed in Python programming language, is planned to be used not only in this Master Thesis but also in further educational and research projects in the frame of simulated robotics. Therefore, the simulator developed in this Master Thesis constructs the basis of a simulator that merges swarm robotics, neuronal dynamics and evolutionary computation.

1.3 Document layout

The remaining chapters of this Master Thesis are structured as follows. Chapter 2 provides theoretical notions on CTRNNs and its underlying neuron models. The explanation is initiated from the simplest level, where the model and dynamics of isolated point neurons are stated. Subsequently, at the network level, the CTRNN is exposed both from an architectural perspective, as directed graph of connected neurons, and from a dynamical perspective, treating the CTRNN as a non linear dynamical system. The higher level is exposed using attractor theory in dynamical systems as a tool for explaining working memory and CTRNN functioning. In addition to CTRNNs, the evolutionary computation algorithms used in this project, namely, GA and SNES, are also described. GA data flow and operators are explained, focusing on the ones employed in the experiments. Alternatively, the theory of NES family of algorithms is explained, ending with a description of SNES as a suitable algorithm for evolving ANNs.

Chapter 3 explains the robotics simulator mechanics from a mathematical perspective. In addition to the environment model, the robot driving system, sensors and actuators are described. The designed communication means that the agents can harness to cooperate is also established in detail. Moreover, the chapter provides an overview of the neural controller and the GA and NES algorithms, highlighting the hyperparameters that are common in all the experiments.

In contrast to Chapter 3, Chapter 4 describes the designed and developed software simulator. More precisely, an explanation of the overall simulator structure, composed by stacked layers and information exchange interfaces is provided. Additionally, other features such as the configuration files, used to create and automatize experiments, the ANN data flow and the parallelization of the execution are also exposed. This chapter ends with an enumeration of several features to be implemented in future versions of the software.

Chapter 5 settles the mechanics of all the experiments in this Master Thesis, namely, leader selection, borderline identification, orientation consensus and light follower. Each experiment is split into the following functional building blocks: (i) description of the task, (ii) statement of the precise sensors, actuators and communication context to be used, (iii) specification of the precise neural architecture, (iv) description of related works in the literature and, lastly, (v) presentation and explanation of the fitness function designed for each task.

Chapter 6 builds on top of the previous chapter by analyzing the evolved behaviors and results of the tasks. In each experiment, a comparison between GA and SNES in terms of fitness scores is revealed. Thereafter, we focus on the analysis of the results of the most relevant solution, either in terms of fitness outperformance or emerged behavior complexity. The overall analysis is fragmented into behavior analysis, scalability assessment, robustness validation and emerged communication analysis.

Chapter 7 concludes the Master Thesis, highlighting the most relevant results and comparing the different communication that emerged in the experiments. Moreover, several future lines of research are presented. Finally, Appendix A displays an example of a

configuration file used to specify the setups of each experiment. Appendix B describes the potential impact that this Master Thesis can have in terms of social, economical, ethical and environmental aspects. Moreover, Appendix C exposes the economic budget of this Master Thesis.

Chapter 2

Theoretical Background

In this chapter, a description of the mathematical concepts and theory used in this Master Thesis is provided. The chapter is divided into two differentiated parts. Firstly, a detailed explanation on the continuous-time recurrent neural networks is considered. Specifically, we start exposing the neuron and the mathematical model used to describe its dynamics, as the most basic neural unit. Subsequently, the neural network is explained from two coexisting perspectives, namely, as a directed graph and as a non-linear dynamical system. Finally, the dynamics of continuous-time recurrent neural networks are visualized from a higher level point of view by studying and analyzing dynamical system attractors and their relevance in learning and working memory formation.

The second part of the chapter shows the theory underlying the evolutionary computation algorithms used in this Master Thesis. More precisely, genetic algorithms and natural evolution strategies are exposed. In the case of genetic algorithms, the different steps involved in the population updates of the evolution process are described. Moreover, the precise operators used in the genetic algorithm are presented in detail. Apart from genetic algorithms, natural evolution strategies are employed in the experiments. Specifically, the theoretical background of separable natural evolution strategies is provided.

2.1 Continuous-time recurrent neural networks

Recurrent Neural Networks (RNNs) are a class of artificial neural networks capable of retaining past events by feeding neuron outputs back through recurrent connections. In constrast to discrete-time RNNs [64] and its variants [65, 66], Continuous-Time Recurrent Neural Networks (CTRNNs) [41] are a type of RNNs with dynamics, stimuli and outputs operating in continuous time. In problems and applications that are intrinsically continuous in time, such as robotics, CTRNNs are of much more interest than their discrete-time counterparts. Clearly, CTRNNs are executed in computers and, thus, their simulations require temporal discretization at some extent. Nevertheless, as their dynamical systems are numerically solved using, e.g., Euler method with a sufficiently small Euler step, their continuous-time characteristic is approximately preserved.

Hereafter, CTRNNs are described starting from their most basic part, namely the neuron and the employed neuron model. Subsequently, the overall neural network is explained from two different coexisting perspectives. Finally, the concept of attractor, from dynamical systems theory, is considered in the context of CTRNNs as a higher level description of the neuronal dynamics.

2.1.1 The neuron model

As a first approach to CTRNNs, we model the dynamics of a single neuron using the firing rate model. Firing rate models (see Chapter 11 in [67]) simplify the functioning of biologically plausible spiking neural networks assuming rate coding schemes. Opposite to the ANN models commonly used in machine learning and deep learning, spiking neuron models [68, 67] are dynamical systems that generate all or none outputs known as spikes or action potentials. Thus, the exchange of information among neurons is not directly produced by the magnitude of the outputs, as in machine learning neural networks. On the contrary, two of the most used theoretical neural codes state that the information resides in the precise timing of spike events (temporal coding) or in the rate at which spikes are generated (rate coding), see Chapter 1 of [68] or [69].

Assuming rate coding, firing rate models utterly simplify spiking neurons by using a non-linear mapping of the membrane voltage directly to the firing rate of a neuron or a population of neurons, instead of returning single spikes. The dynamics of the firing rate model are shown in Eqs. (2.1) and (2.2)

$$\tau \frac{dv(t)}{dt} = -v(t) + \mathrm{I}(t) \tag{2.1}$$

$$u(t) = F(g(v(t) + \beta)) \tag{2.2}$$

Let us describe the above equations. Inspired from biological neuronal dynamics, $v(t)$ represents the time varying membrane potential of the neuron and $\mathrm{I}(t)$ is the instantaneous somatic current injected in the neuron as the contribution from all presynaptic neurons. Additionally, τ is the membrane decaying time constant governing how rapidly $v(t)$ reaches stable fixed points in response to $\mathrm{I}(t)$. The second equation exposes the transformation of the membrane voltage into the variable $u(t)$ representing the neuron firing rate or activity. F is a non-linear function that accomplishes the mentioned mapping. The most common non-linearities, and the ones considered in this project, are the sigmoid function (see Eq. 2.3) and the hyperbolic tangent function (see Eq. 2.4).

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{2.3}$$

$$\tanh(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}} \tag{2.4}$$

Besides, β and g are constants that define neuron dynamical properties. In the case of β, it establishes the maximum voltage threshold that must be surpassed in order to produce non-zero firing rate $u(t)$. This property mimics how action potentials are, in a simplified way, produced in spiking neuron models when the membrane voltage surpasses some maximum threshold value. The constant g states how fast a neuron can switch from resting activity (null or scarce activity) to maximum firing rate. Note that large values will increase the slope of the sigmoid function within the linear region, producing a fast transition between minimum and maximum activities.

2.1.2 The neural network

In order to describe the CTRNN, we consider two different and coexisting frameworks. On the one hand, from an architectural point of view, the neural network is understood as a weighted directed graph

$$\mathcal{G}_{\mathrm{RNN}}(\mathcal{V}, \mathcal{E}) \tag{2.5}$$

where $\mathcal{V}$ is the set of ordered neurons in the network and $\mathcal{E}$ comprises the set synapses or connections between neurons. For a reinforced notation, $\mathcal{V}$ is partitioned as

$$\mathcal{V} = \mathcal{N}_{\mathcal{I}} \cup \mathcal{N}_{\mathcal{H}} \cup \mathcal{N}_{\mathcal{M}}$$

where the subsets $\mathcal{N}_{\mathcal{I}}$, $\mathcal{N}_{\mathcal{H}}$ and $\mathcal{N}_{\mathcal{M}}$ represent input, hidden and motor neurons respectively. Note that $\mathcal{N}_{\mathcal{I}}$ elements are not actual neuron models but, instead, they are node placeholders in the graph representing input stimuli signals. The subsets $\mathcal{N}_{\mathcal{H}}$ and $\mathcal{N}_{\mathcal{M}}$ are composed by firing rate models with their own dynamics, as it was previously explained. The set of non input neurons is introduced as $\mathcal{N} = \mathcal{N}_{\mathcal{H}} \cup \mathcal{N}_{\mathcal{M}}$. Thus, in the following, by I we refer to the cardinality of $\mathcal{N}_{\mathcal{I}}$ representing the dimension of the input stimuli tuple. Similarly, N will denote the cardinality of $\mathcal{N}$ as the number of firing rate neurons. Let $\mathbf{W} \in \mathbb{R}^{(N+I)\times(N+I)}$ be the weighted adjacency matrix of $\mathcal{G}_{\text{RNN}}$. Notice that the first I rows of $\mathbf{W}$ are zeros since input nodes have zero in-degree. For notation convenience, the adjacency matrix is partitioned into submatrices as follows:

$$\mathbf{W} = \left(\begin{array}{c|c} \mathbf{O}_{I\times I} & \mathbf{O}_{I\times N} \\ \hline \mathbf{W}_{\mathcal{I}} & \mathbf{W}_{\mathcal{N}} \end{array}\right)$$

where $\mathbf{W}_{\mathcal{I}} \in \mathbb{R}^{N\times I}$ is the submatrix comprising connections from input nodes in $\mathcal{N}_{\mathcal{I}}$ to neurons in $\mathcal{N}$ and $\mathbf{W}_{\mathcal{N}} \in \mathbb{R}^{N\times N}$ describes edges among neurons in $\mathcal{N}$. $\mathbf{O}_{I\times I}$ and $\mathbf{O}_{I\times N}$ are the zero matrices of dimension given by the subindex. Up to this point, CTRNN topologies or architectures can be fully described by $\mathcal{G}_{\text{RNN}}$. However, its dynamics, that are not fully defined yet, will be treated below.

As an example, consider the CTRNN in Fig. 2.1 with $I = 2$, and $N = 3$. The sets $\mathcal{N}_{\mathcal{I}}$, $\mathcal{N}_{\mathcal{H}}$ and $\mathcal{N}_{\mathcal{M}}$ are colored in yellow, blue and red. The weights of the edges are attached to each connection arrow and the ordering of the neurons is inside the vertices.

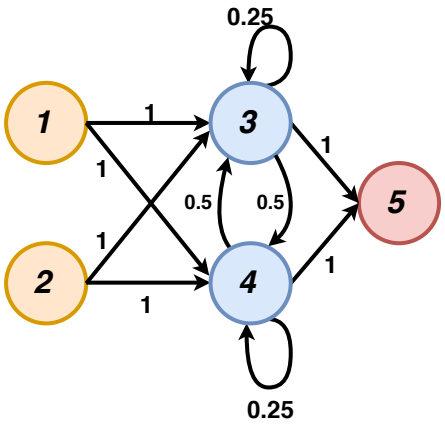


Figure 2.1: Example of a simple CTRNN with 2 inputs (yellow), 2 hidden neurons (blue) and 1 motor neuron (red).

For this example, the corresponding adjacency matrix is:

$$\mathbf{W} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0.25 & 0.5 & 0 \\ 1 & 1 & 0.5 & 0.25 & 0 \\ 0 & 0 & 1 & 1 & 0 \end{pmatrix}$$

and, thus, the submatrices $\mathbf{W}_{\mathcal{I}}$ and $\mathbf{W}_{\mathcal{N}}$ are:

$$\mathbf{W}_{\mathcal{I}} = \begin{pmatrix} 1 & 1 \\ 1 & 1 \\ 0 & 0 \end{pmatrix}, \quad \mathbf{W}_{\mathcal{N}} = \begin{pmatrix} 0.25 & 0.5 & 0 \\ 0.5 & 0.25 & 0 \\ 1 & 1 & 0 \end{pmatrix}$$

In addition to the graph theory perspective, the CTRNN can be seen as an N-dimensional non linear dynamical system. Let $\mathbf{v}(t) = (\mathbf{v}_1(t), \ldots, \mathbf{v}_N(t))^\top$ be the instantaneous membrane voltage vector of the N neurons in $\mathcal{G}_{\text{RNN}}$ and $\boldsymbol{\phi}(t) = (\boldsymbol{\phi}_1(t), \ldots, \boldsymbol{\phi}_I(t))^\top$ the stimuli vector being fed to the network. Thereafter, the dynamics of a CTRNN can be fully described as an N-dimensional non-linear dynamical system defined by Eq. 2.6.

$$\boldsymbol{\tau} \odot \frac{d\mathbf{v}(t)}{dt} = -\mathbf{v}(t) + \mathbf{W}_{\mathcal{N}}\, \mathbf{F}(\mathbf{g} \odot (\mathbf{v}(t) + \boldsymbol{\beta})) + \mathbf{W}_{\mathcal{I}}\, \boldsymbol{\phi}(t) \tag{2.6}$$

where $\mathbf{W}_{\mathcal{N}}$ and $\mathbf{W}_{\mathcal{I}}$ are the submatrices of the adjacency matrix of $\mathcal{G}_{\text{RNN}}$ as described above. $\boldsymbol{\tau} = (\boldsymbol{\tau}_1, \ldots, \boldsymbol{\tau}_N)^\top$ is the vector of neuron time constants, $\boldsymbol{\beta} = (\boldsymbol{\beta}_1, \ldots, \boldsymbol{\beta}_N)^\top$ is the vector of neuron biases and $\mathbf{g} = (\mathbf{g}_1, \ldots, \mathbf{g}_N)^\top$ is the tuple of neuron gains. $\boldsymbol{\beta}$ and $\mathbf{g}$ define the firing behavior and nature of neurons as already seen in the previous subsection. $\mathbf{F}(\mathbf{z})$ is defined as the element-wise function applied to vector $\mathbf{z}$ (see Eq. 2.7).

$$\mathbf{F}(\mathbf{z}) = \begin{pmatrix} F(\mathbf{z}_1) \\ \vdots \\ F(\mathbf{z}_N) \end{pmatrix} \tag{2.7}$$

F is commonly the sigmoid or hyperbolic tangent functions (Eqs. 2.3 and 2.4). $\odot$ is the element-wise multiplication.

Additionally, the vector of firing rates or activities is generalized from Eq. 2.2 as follows:

$$\mathbf{u}(t) = \mathbf{F}(\mathbf{g} \odot (\mathbf{v}(t) + \boldsymbol{\beta})) \tag{2.8}$$

Eq. 2.8 is not included in the system of non-linear equations in (2.6) with the aim of representing the CTRNN dynamics as compact as possible. Nevertheless, $\mathbf{u}(t)$ plays a crucial role in the CTRNN because, apart from being the variable to be decoded into agent actions, it will be used to represent state trajectories in the bounded state space $[0, 1]^N$ (see Subsection 2.1.3). Finally, in order to iteratively solve the system of differential equations in 2.6, Euler method is used. Euler method with a step size Δt leads to the state updates in Eq. 2.9. In this work, the initial state is always assumed to be $\mathbf{v}(0) = (0, \ldots, 0)^\top$.

$$\begin{cases} \mathbf{v}(t + \Delta t) = \mathbf{v}(t) + \tilde{\boldsymbol{\tau}} \odot (-\mathbf{v}(t) + \mathbf{W}_{\mathcal{N}}\, \mathbf{F}(\mathbf{g} \odot (\mathbf{v}(t) + \boldsymbol{\beta})) + \mathbf{W}_{\mathcal{I}}\, \boldsymbol{\phi}(t)) \\ \mathbf{u}(t + \Delta t) = \mathbf{F}(\mathbf{g} \odot (\mathbf{v}(t + \Delta t) + \boldsymbol{\beta})) \\ \tilde{\boldsymbol{\tau}}_i = \dfrac{\Delta t}{\boldsymbol{\tau}_i},\ \forall i \in \{1, \ldots, N\} \end{cases} \tag{2.9}$$

2.1.3 Attractors

Considering the dynamics of the CTRNN presented in Eq. 2.6, we describe the concept of attractors in dynamical systems. Given an N-dimensional system of differential equations, an attractor is a subset, contained in the state or phase space, that is the steady state solution of the dynamical system for all initial conditions in a neighborhood of the attractor. More precisely, an attractor $\mathfrak{A}$ (see Chapter 13 of [70]) is a manifold of dimension, say K, lower than the dimension of the state space $[0, 1]^N$ and contained in it. Note that each component of the state space is constrained to $[0, 1]$, assuming that the non-linearity is the sigmoid activation, as

we are using the firing rate state space instead of the voltage state space $\mathbb{R}^N$ (see the mapping between these vector spaces in Eq. 2.8). The attractor $\mathfrak{A}$ is then the manifold embedded in the state space of the CTRNN and defined by a finite set of equations; say

$$\mathfrak{A} \equiv \left\{ \begin{array}{c} M_1(u_1, \ldots, u_N) = 0 \\ M_2(u_1, \ldots, u_N) = 0 \\ \vdots \\ M_K(u_1, \ldots, u_N) = 0 \end{array} \right\} \tag{2.10}$$

For a better understanding on the concept of attractor, the basin of attraction is also described. The basin of attraction $B(\mathfrak{A})$ of $\mathfrak{A}$ is a neighborhood of the attractor so that any initial state belonging to the basin will eventually converge to $\mathfrak{A}$. It should be mentioned that, in the context of CTRNNs, the previous consequence of being in the basin of attraction of some attractor can be avoided if a sufficiently abrupt change in the input stimuli leads to $\mathbf{u} = (u_1, \ldots, u_N)^\top$ escaping from the basin of attraction region. In such a case, $\mathbf{u}$ would fall in the basin of attraction of another attractor $\mathfrak{A}'$. That is because all the basins of all the attractors in a dynamical system form a partition of the state space. However, for the moment, it is assumed that the dynamical system is autonomous. The definition of $\mathfrak{A}$ can be formalized as a subset of the state space $[0, 1]^N$ fulfilling the following conditions:

1. If $\mathbf{u}(t) \in \mathfrak{A} \Rightarrow \mathbf{u}(t + T) \in \mathfrak{A} \ \ \forall T \in [0, \infty)$
2. If $\mathbf{u}(t) \in B(\mathfrak{A}) \Rightarrow \mathbf{u}(t + T) \in B(\mathfrak{A}) \ \ \forall T \in [0, \infty)$
3. If $\mathbf{u}(t) \in B(\mathfrak{A}) \Rightarrow \lim_{t \to \infty} \mathbf{u}(t) \in \mathfrak{A}$

There are multiple kinds of possible attractors of a dynamical system. The most common type of attractor is called point attractor, and it is caused by stable fixed points in the system. In contrast to point attractors, the attractor is a limit cycle when the trajectory of the state inside $\mathfrak{A}$ is a periodic trajectory for some period T. That is, provided that $\mathbf{u}(t) \in \mathfrak{A}$, then

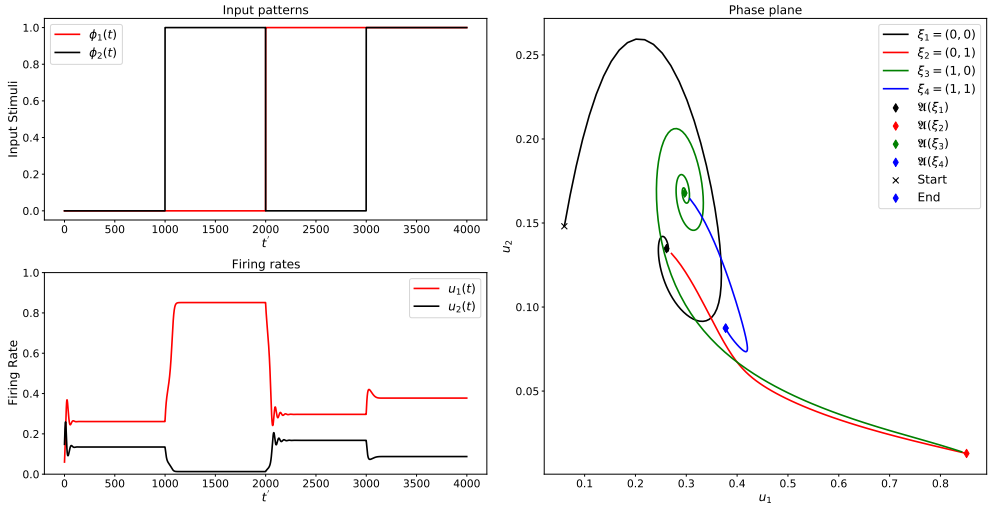
$$\mathbf{u}(t + T) = \mathbf{u}(t) \text{ for some } T > 0$$

Additionally, attractors can be a line, a torus or chaotic attractors, among other possibilities.

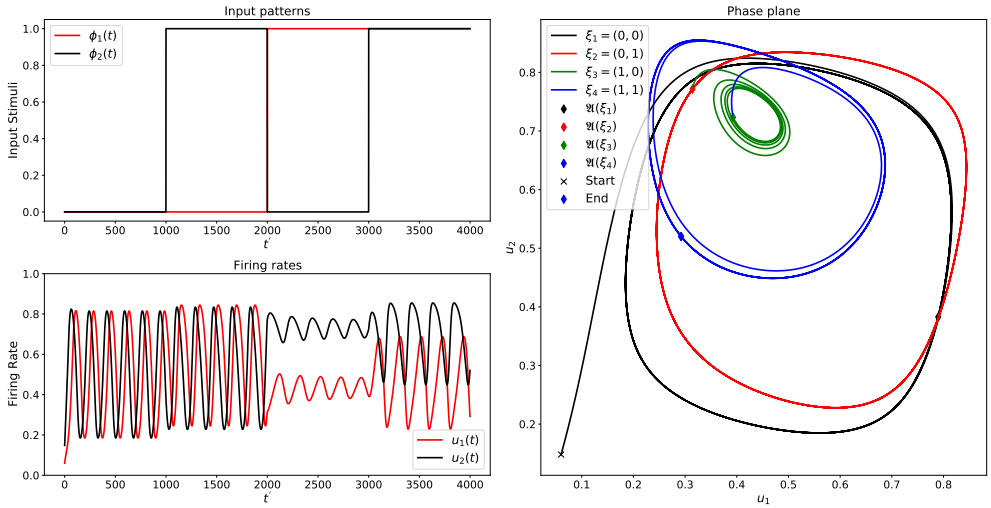
Up to this point, attractors have been described considering that the input stimuli $\boldsymbol{\phi}(t) = 0 \ \forall t$. Therefore, the attractor towards which the state trajectory tends can be fully determined given an initial state $\mathbf{u}(0)$. More interestingly, we now describe the scenario where the input of the system can be non-zero. In this case, the location or even the type of attractors are altered in response to $\boldsymbol{\phi}(t)$. Moreover, given that $\mathfrak{A}$ is the attractor towards which the system evolves at some point in the trajectory, and that the CTRNN parameters lead to multiple attractors coexisting in the state space, a large enough stimuli variation $\frac{d\boldsymbol{\phi}(t)}{dt}$ can cause $\mathbf{u}(t)$ to escape from $B(\mathfrak{A})$. Figure 2.2 exemplifies the concept of attractor displacement or alteration. It shows the state trajectories of three different two neuron CTRNNs when different input patterns are injected. In all the cases, the bias vector is set to $\boldsymbol{\beta} = (-2.75, -1.75)^\top$, the gains are all fixed to 1 and the membrane time constants are $20dt$. Additionally, the employed adjacency matrices $\mathbf{W}$ are:

$$\mathbf{W}_a = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 5 & 3 \\ 1 & 1 & -3 & 5 \end{pmatrix} \quad \mathbf{W}_b = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ -0.2 & 0 & 4.5 & 1 \\ -0.1 & 0.1 & -1 & 4.5 \end{pmatrix} \quad \mathbf{W}_c = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & -2, & 4.5 & 1 \\ -1 & 1 & -1 & 4.5 \end{pmatrix}$$

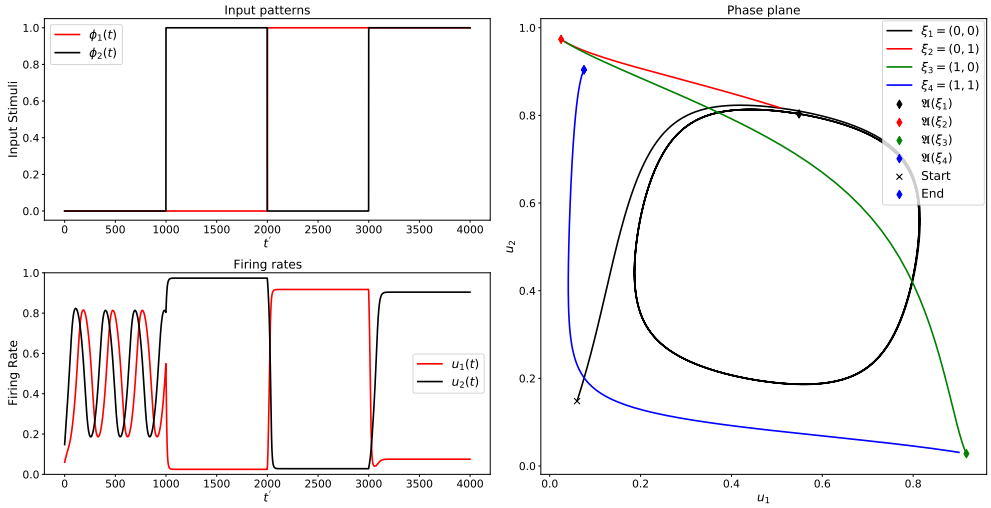
The input patterns $\boldsymbol{\phi}$ presented sequentially are $(0, 0)$, $(0, 1)$, $(1, 0)$ and $(1, 1)$, as exposed in the upper left plots. The firing rates $\mathbf{u}(t)$ generated in response to the patterns are exposed in



(a)



(b)



(c)

Figure 2.2: Three experiments of the impact of external input $\boldsymbol{\phi}(t) \in \{0, 1\}^2$ on two neuron CTRNNs. The upper left plots of (a), (b) and (c) show the temporal variation of the components of $\boldsymbol{\phi}(t)$. The lower left plots are the output firing rates $\mathbf{u}(t)$ of the neurons. On the right sides, state planes with the given trajectories are depicted (colors represent present stimuli). (a) Input patterns change the position of point attractor. (b) Input patterns alter the limit cycle attractor. (c) Limit cycle is transformed into point attractor for different input patterns.

the lower left plots. The right graphics depict the state plane and the corresponding trajectory in each of the a, b and c scenarios. Colors represent the stimuli pattern that is being injected into the network at a precise point in the state trajectory. In Fig. 2.2a, changes in $\boldsymbol{\phi}$ result in a translation of the point attractor in the state plane. Fig. 2.2b shows a dynamical system in which the attractor is a limit cycle being altered by input variations. Finally, in Fig. 2.2c, the attractor for zero input is a limit cycle that is transformed into a point attractor for different patterns of $\boldsymbol{\phi}$.

In general, for a dynamical system driven by an external signal, such as in the CTRNN, the notation for describing attractors can be modified as $\mathfrak{A}(\boldsymbol{\phi})$, for any $\mathfrak{A}$ in the system, to emphasize the dependency of $\mathfrak{A}$ with respect to $\boldsymbol{\phi}$. In this way, the stimuli vector $\boldsymbol{\phi}$ can be treated as a parameter that modulates the steady states of the system. For a CTRNN with two neurons and a single input $\phi(t)$ we can analyze how the parameter ϕ bounded in, for example $[0, 1]$, alters the attractor of the network. Figure 2.3 depicts this experiment for two different two-neuron neural networks. The parameter ϕ is varied from zero to one with steps of 0.1 in the first plot and 0.05 in the second one. For each value of ϕ, the trajectory of the state is recorded using Euler method with the same initial condition in all trials and with enough simulation time steps to guarantee convergence. The color of the trajectories denotes the value of ϕ for that trajectory, being constant in all their duration. Consequently, in Fig. 2.3a

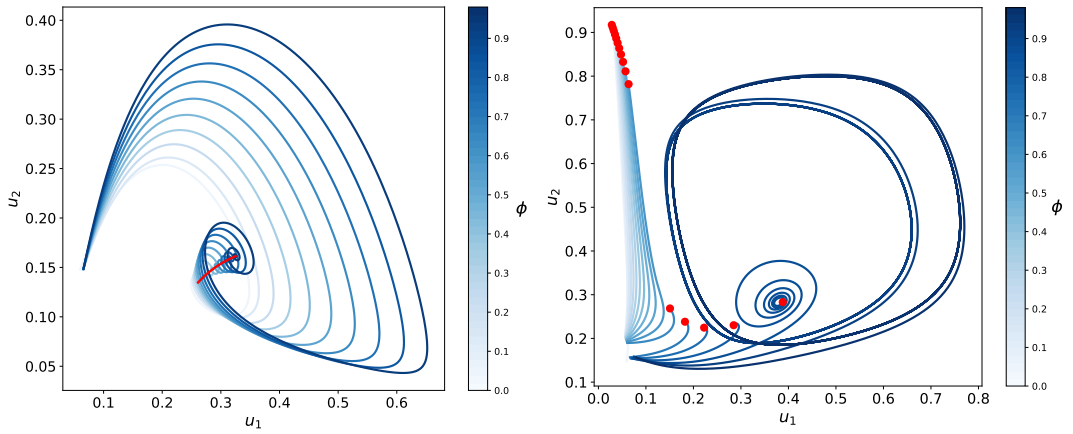


Figure 2.3: Impact of input stimuli ϕ on the attractor of two different CTRNNs with 2 neurons. (a) Point attractor is translated along a curve. (b) A Hopf bifurcation transforms point attractor into a limit cycle.

it can be observed that ϕ moves the point attractor along a curve (in red) in the state space. A possible interpretation of this fact is that, although the attractor is a point when evaluated for some input value, the general attractor for any value of ϕ is a curve attractor. More interestingly, in the dynamical system of Fig. 2.3b, we show a Hopf bifurcation. It is a Hopf bifurcation because there is a critical ϕ_c (somewhere in between 0.9 and 0.95) that causes the transformation of a point attractor (red dots) into a limit cycle, as shown in the graphic.

Now, we return to the general case of an N-dimensional CTRNN with dynamics fixed by $\mathbf{W}$, $\mathbf{g}$ and $\boldsymbol{\beta}$ and modulated by the vector $\boldsymbol{\phi}(t)$. Suppose that the system has, at most, T attractors $\{\mathfrak{A}_1(\boldsymbol{\phi}), \ldots, \mathfrak{A}_T(\boldsymbol{\phi})\}$ for every $\boldsymbol{\phi}$ and that the external dynamics of $\boldsymbol{\phi}(t)$ work at a much slower time scale k. This means that the trajectory of the CTRNN state has enough time to converge to an attractor, or at least to remain in a neighborhood of the attractor of much less area than its basin (depending on $\boldsymbol{\tau}$ of the CTRNN), before $\boldsymbol{\phi}$ varies again. For

a given initial attractor $\mathfrak{A}(0, \boldsymbol{\phi}(0)) \in \{\mathfrak{A}_1(\boldsymbol{\phi}(0)), \ldots, \mathfrak{A}_T(\boldsymbol{\phi}(0))\}$ and a sequence of external stimuli $\{\boldsymbol{\phi}(k)\}_{k=0}^{\infty}$, there is a sequence of attractors as follows:

$$\{\mathfrak{A}(k, \boldsymbol{\phi}(k))\}_{k=0}^{\infty} \tag{2.11}$$

Provided that the attractors are wisely optimized to store meaningful patterns of $\boldsymbol{\phi}$ or a transformation of $\boldsymbol{\phi}$ in some latent space, the sequence 2.11 of attractors encodes the sequence of stimuli patterns provided as input in the past. For instance, patterns of light intensity signal can discretize values representing very low, low, medium, high and very high light intensity. This sequence of attractors and their transient heteroclinic trajectories play an important role in working memory [71, 72, 73, 74] among other cognitive tasks. Artificial recurrent neural networks that harness and optimize the attractors of the system, their stored patterns and the heteroclinic transitions among attractors are commonly denoted as attractor neural networks. One of the most important contributions in this line is the Hopfield neural network [75], that is a content addressable memory of input patterns so that, once optimized, whole patterns can be retrieved from incomplete input samples. Additionally, [76] describes a general framework for controlling attractor networks based on spiking neural networks.

2.2 Neuroevolution algorithms

Neuroevolution is the application of evolutionary computation algorithms to optimize the parameters of artificial neural networks, leading to large dimensional optimization problems (see [38]). There are multiple variants of neuroevolutionary algorithms, depending on the evolutionary algorithm used, the genotype representation employed or the neural network model to be evolved. Just to mention some of them, in [7, 12, 39, 40, 42, 43] ANNs are evolved using genetic or evolutionary algorithms. Cooperative coevolutionary algorithms are proposed in [77, 78] as optimizers of ANNs leading to multiple subpopulations of neural parameters being evolved in parallel. Moreover not only the weights of the synapses but the neural topology itself is evolved in [79] and its extension devoted to deep neural architectures [80]. As a final example, evolution strategies have been also explored in the context of ANN optimization [81, 82]. This Master Thesis focuses on genetic algorithms and natural evolution strategies as neuroevolution algorithms used to evolve CTRNN models in multi-agent robotics tasks.

2.2.1 Genetic Algorithms

Genetic algorithms (GA) are biologically inspired population based optimization algorithms that mimic how natural selection and survival of the fittest work in nature (see [45]). A population of candidate solutions, namely individuals, genotypes or chromosomes, are iteratively updated with the aim of maximizing some performance score defined by a fitness function. Using the evaluated fitness value associated to each genotype, a set of genetic operators are sequentially applied to the overall population in order to generate the population of the next generation or iteration of the GA. A canonical GA is composed by the following operators:

1. *Selection*: given the fitness values of the genotypes, the selection operator is responsible of selecting the subset of individuals in the population that are used as parents to generate offspring in subsequent operators. The selected genotypes can be deterministically chosen using their fitness values or it can be a stochastic selection using fitness scores to build probabilities.
2. *Crossover*: or recombination operator, uses the subset of selected parents to generate new offspring individuals that will constitute the next population. Firstly, through a

mating strategy, that in this case is a random mating, the genotypes are, generally, pairwise grouped. Thereafter, the alleles of each parent of the pair are combined or merged in some way to produce the chromosome of two new individuals. It should be mentioned that there is a parameter called crossover probability or crossover rate p_{xvr} representing the probability that two parents produce offspring. In absence of descendants, the parents are directly added to the offspring set.

3. *Mutation*: The offspring genotypes, resulting from the recombination phase, are subject to a mutation step that, with some mutation probability p_{mut}, alters the genes of the individuals. Mutation operators are mainly used as mechanisms for exploring new areas of the search space and avoid premature convergence of the GA.

4. *Elite preservation*: a small subset of the population genotypes before crossover and mutation are directly selected as individuals of the new generation population. These elite genotypes are selectively picked in accordance with the fitness values. That is, only the individuals with highest fitness are deterministically selected.

In this Master Thesis, the population of a genetic algorithm is defined as $\mathcal{P}(g) = \{\mathscr{G}_1, \ldots, \mathscr{G}_\lambda\}$. It depends on the current generation $g \in \mathbb{Z}_{\geq 0}$, and it is a set containing λ genotypes (λ is usually called the population size), where $\mathscr{G}_i$ is a vector. Even though there are multiple genotype representations, recurrent neural networks are encoded using a fixed length vector of the network optimization parameters (synapse strengths, neuron time constants, biases and gains). More precisely, given that d is the search space dimension, the genotype vectors as expressed as:

$$\mathscr{G}_i = (\mathscr{G}_{i,1}, \ldots, \mathscr{G}_{i,d})^\top,\ i \in \{1, \ldots, \lambda\}$$

Opposed to the genotype, representing the set of genes of an individual, the phenotype is the set of developed characteristics of an individual at the evaluation level, decoded from its corresponding genotype. For instance, in neuroevolution problems with direct genotype representations (fixed length tuples) the genotype is the vector composed by all the synapse strengths and the phenotype is the ANN model.

Besides, the fitness value of a genotype is defined as the result of the following function,

$$\begin{array}{rccc} f: & \mathbb{R}^d & \longrightarrow & \mathbb{R} \\ & \mathscr{G} & \longmapsto & F \end{array}$$

The precise function depends on each particular application and will be exposed, task by task, in Chapter 5. In fields such as robotics or reinforcement learning, the fitness evaluation is costly obtained by simulating the task during an episode of length T_E. In these cases, the computed fitness is an estimator of the true unknown expected fitness of the genotype using the sample mean of sample size equal to the number of evaluation trials (or episodes). There must be a tradeoff between variance reduction of the estimation and computational cost reduction. Another important aspect to be pointed out is the initialization of the population genotypes at $g = 0$. The genes are randomly initialized according to a uniform distribution with higher and lower user defined bounds of the CTRNN parameters (see Chapter 3). Fig. 2.4 exposes the diagram of the canonical GA with the mentioned steps. The stopping criteria can be either reaching a solution whose fitness distance to the optimal solution F^* is lower than a small value ϵ or fixing a maximum number of generations G (or the combination of both). In the remaining part of this section, the precise genetic operators to be used are described.

Selection operator: Tournament Selection

Tournament selection is used as selection operator in this Master Thesis (see [83]). Let $\mu \in \{1, \ldots, \lambda\}$ be the number of genotypes to be selected as parents. In tournament selection,

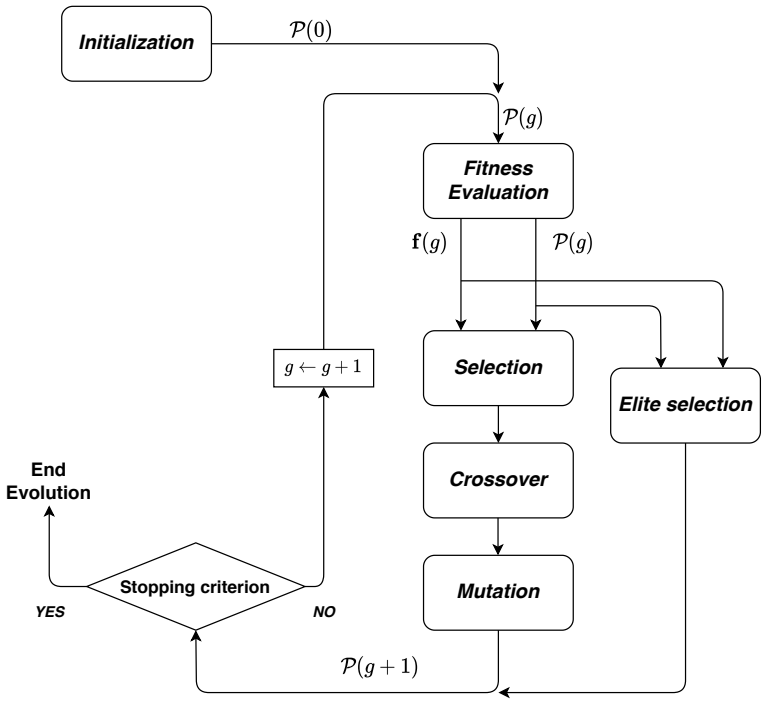


Figure 2.4: Diagram of canonical genetic algorithm.

μ different groups of K individuals are randomly chosen from the current population. Every individual has the same probability to be in a tournament, regardless of their estimated fitness. For each group or tournament, a winner is established as the genotype in the group with highest fitness. The set of winners from all tournaments are gathered to form the set of selected genotypes $\mathcal{P}_{sel} \subset \mathcal{P}$. Algorithm 1 summarizes tournament selection process in GAs.

Algorithm 1: Tournament Selection

input: $\mathcal{P}$, $f(\mathscr{G})\ \forall \mathscr{G} \in \mathcal{P}$, K
output: $\mathcal{P}_{sel}$
for $i = 1, \ldots, \mu$ **do**
  Select randomly a tournament of K individuals $T_i = \{\mathscr{G}_k\}_{k=1}^{K}$ from $\mathcal{P}$
  Set tournament winner as the genotype with highest fitness $\underset{\mathscr{G} \in T_i}{\operatorname{argmax}}\{f(\mathscr{G})\}$
  Add winner to selected individuals set $\mathcal{P}_{sel}$
end

Tournament selection is suitable for reducing premature convergence chances as it reduces selective pressure (if tournament size K is sufficiently small).

Crossover operator: BLX-α

BLX-α, as the crossover operator used, selects offspring chromosomes stochastically in the neighborhood of the parents (see [84, 85]). Let $\mathscr{G}_a$ and $\mathscr{G}_b$ be two population individuals to be recombined to generate two novel offspring genotypes. Additionally, let vectors $\mathscr{G}_{min}$, $\mathscr{G}_{max} \in \mathbb{R}^d$ be the gene-wise minimum and maximum vectors of the selected parents, whose components are defined as:

$$\mathscr{G}_{min,i} = \min\{\mathscr{G}_{a,i}, \mathscr{G}_{b,i}\}, \quad \mathscr{G}_{max,i} = \max\{\mathscr{G}_{a,i}, \mathscr{G}_{b,i}\}, \quad \forall i \in \{1, \ldots, d\}$$

The hyperrectangle C is constructed as:

$$C = \prod_{i=1}^{d} [\mathbf{c}_i, \mathbf{d}_i] = [\mathbf{c}_1, \mathbf{d}_1] \times \cdots \times [\mathbf{c}_d, \mathbf{d}_d]$$

where $\mathbf{c}$ and $\mathbf{d}$ are computed as follows,

$$\mathbf{c} = \mathscr{G}_{min} - \alpha \, |\mathscr{G}_a - \mathscr{G}_b|$$

$$\mathbf{d} = \mathscr{G}_{max} + \alpha \, |\mathscr{G}_a - \mathscr{G}_b|$$

where α is a fixed hyperparameter that establishes the area of exploration. Thereafter, the offspring genotypes $\mathscr{G}_a^{'}$ and $\mathscr{G}_b^{'}$, resulting from crossover, are uniformly sampled from C:

$$\mathscr{G}_a^{'} = (u_1, \ldots, u_d)^\top, \; \mathscr{G}_b^{'} = (v_1, \ldots, v_d)^\top \mid u_i, v_i \sim \mathcal{U}(\mathbf{c}_i, \mathbf{d}_i), \; \forall i \in \{1, \ldots, d\}$$

Note that more than two children can be obtained by sampling new genotypes. By exploration

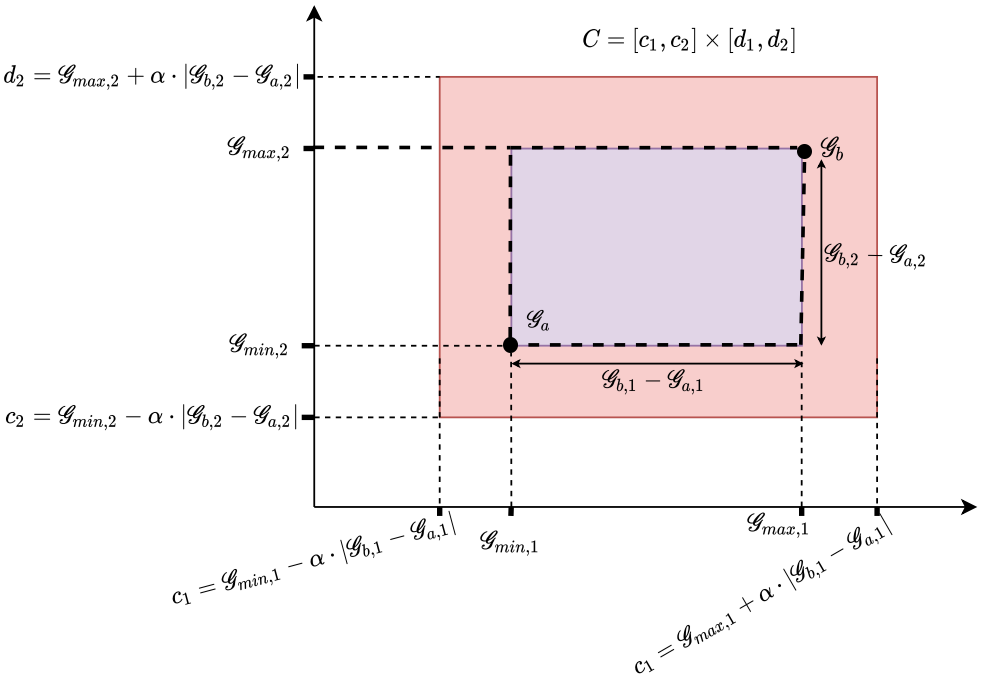


Figure 2.5: Example of a BLX-α sampling rectangle with $d = 2$.

and exploitation regions we denote the following. Let C_{xpt} be another hyperrectagle, defining the exploitation area,

$$C_{xpt} = [\mathscr{G}_{min,1}, \mathscr{G}_{max,1}] \times \cdots \times [\mathscr{G}_{min,d}, \mathscr{G}_{max,d}]$$

so that C_{xpt} is equal to C when $\alpha = 0$, resulting in a non-exploration recombination. Sampled genotypes belonging to C_{xpt} exploit the known parents' alleles, increasing genetic drift. On the contrary the exploration region is $C_{xpl} = C \setminus C_{xpt}$, whose points promote exploration of new solutions and population diversity. As an example, Fig. 2.5 shows the sampling area of BLX-α for a search space of dimension $d = 2$. The exploitation area is colored in purple and the exploration area is the red region.

Mutation operator: Gaussian mutation

Mutation operators allow slight alterations on the genotypes so that unexplored regions of the search space can be reached. Mutation probability p_{mut} sets the balance between exploration and exploitation. Gaussian mutation is a common choice in the case of real coded GA. Gaussian mutation essentially adds a normally distributed noise to the genes with some probability p_{mut}. Letting σ^2_{mut} be the mutation noise variance, the gaussian mutation can be also seen as resampling the gene $g \in \mathbb{R}$ from $\mathcal{N}(g, \sigma^2_{mut})$ with probability p_{mut}. Formally, given genotype $\mathscr{G}$ subject to mutation the mutation is expressed as in Eq. 2.12.

$$\mathscr{G}_{mut} = \mathscr{G} + \mathbf{p}_{mut} \odot \boldsymbol{\varepsilon}_{mut} \tag{2.12}$$

where $\boldsymbol{\varepsilon}_{mut} \sim \mathcal{N}(0, \sigma^2_{mut}\mathbf{I})$, being $\mathbf{I}$ the identity matrix and $\mathbf{p}_{mut}$ is a boolean mask denoting the genes that are subject to mutation. Its elements are sampled from Bernoulli distribution $\mathcal{B}(p_{mut})$, so that each gene has a probability p_{mut} of mutation in each generation. Finally, $\odot$ represents the element-wise product.

2.2.2 Natural Evolution Strategies

In addition to GA, Natural Evolution Strategies are described in this subsection. Natural Evolution Strategies (NES) are a family of evolutionary computation algorithms that, instead of directly evolving the population of individuals or genotypes, it updates a parametrized Probability Density Function (PDF) that is used to sample genotypes. NES have recently proved to be competitive alternatives to deep reinforcement learning, tested in known benchmark problems such as Atari games or humanoid locomotion(see [82, 81]). In this Master Thesis, Separable Natural Evolution Strategy (SNES) algorithm(see [50]) is used to evolve CTRNN controllers. Thus, in the following, a brief introduction to NES family of algorithms is provided and, in particular, SNES is described at the end of the subsection, building on top of other NES methods. For a simplified description of NES we are following [49]. We refer to this paper for a highly detailed and complete description of NES methods.

Let $J(\boldsymbol{\theta})$ be the expected fitness under genotype generator PDF or search distribution parametrized by $\boldsymbol{\theta}$ and defined as in Eq. 2.13. $\pi(\mathscr{G} \mid \boldsymbol{\theta})$ denotes the mentioned search distribution, $f(\mathscr{G})$ is the fitness function of a genotype and $\mathbb{E}_{\boldsymbol{\theta}}$ means the expected value under a given parametrization.

$$J(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{\theta}}\left[f(\mathscr{G})\right] = \int f(\mathscr{G})\, \pi(\mathscr{G} \mid \boldsymbol{\theta})\, d\mathscr{G} \tag{2.13}$$

The basis of NES is to find the parameters $\boldsymbol{\theta}$ that maximize the expected fitness $J(\boldsymbol{\theta})$. In order to accomplish this task, it uses the natural gradient ascent (NGA) instead of the regular gradient ascent using the plain gradient. Before describing NGA updates, the gradient of the expected fitness with respect to $\boldsymbol{\theta}$ can be conveniently specified as in Eq. 2.14 using the 'log-likelihood trick'.

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{\theta}}\left[f(\mathscr{G})\, \nabla_{\boldsymbol{\theta}} \log(\pi(\mathscr{G} \mid \boldsymbol{\theta}))\right] \tag{2.14}$$

Moreover, the sample estimate of the gradient using λ samples is exposed in Eq. 2.15. Analogous to genetic algorithms, λ represents the population size. Moreover, the population is sampled from the parametrized PDF so that $\mathscr{G}_i \sim \pi(\mathscr{G} \mid \boldsymbol{\theta}),\ \forall i \in \{1, \ldots, \lambda\}$

$$\hat{\nabla}_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \frac{1}{\lambda} \sum_{i=1}^{\lambda} f(\mathscr{G}_i)\, \nabla_{\boldsymbol{\theta}} \log(\pi(\mathscr{G}_i \mid \boldsymbol{\theta})) \tag{2.15}$$

The main difference between NES algorithms and search gradient algorithms, using the plain gradient estimation for gradient ascent, is that NES use the natural gradient (see [86]) estimate for updating the search distribution parameters. The natural gradient can be computed as,

$$\tilde{\nabla}_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathbf{F}^{-1}\, \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \tag{2.16}$$

where $\mathbf{F}$ is the Fisher information matrix of the search distribution defined as in Eq. 2.17.

$$\mathbf{F} = \mathbb{E}\left[\nabla_{\boldsymbol{\theta}} \log(\pi(\mathscr{G} \mid \boldsymbol{\theta}))\, \nabla_{\boldsymbol{\theta}} \log(\pi(\mathscr{G} \mid \boldsymbol{\theta}))^{\top}\right] \tag{2.17}$$

The Fisher information matrix can be estimated from samples as:

$$\hat{\mathbf{F}} = \frac{1}{\lambda} \sum_{i=1}^{\lambda} \nabla_{\boldsymbol{\theta}} \log(\pi(\mathscr{G}_i \mid \boldsymbol{\theta}))\, \nabla_{\boldsymbol{\theta}} \log(\pi(\mathscr{G}_i \mid \boldsymbol{\theta}))^{\top} \tag{2.18}$$

The estimates of $\mathbf{F}$ and $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ lead to the NGA update of the distribution parameters displayed in Eq. 2.19, where $\eta_{\boldsymbol{\theta}}$ is some constant learning rate, and g denotes the current generation or iteration. This parameter update defines the core of the Canonical NES algorithm (see [49]).

$$\boldsymbol{\theta}_{g+1} = \boldsymbol{\theta}_g + \eta_{\boldsymbol{\theta}}\, \hat{\mathbf{F}}_g^{-1}\, \tilde{\nabla}_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_g) \tag{2.19}$$

The updates of the canonical NES algorithm can be utterly costly as the problem dimension increases. It is because of the Fisher information matrix estimation and inverse computation. Particularizing to the multivariate gaussian search distributions $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, exponential natural evolution strategy (xNES) is one of proposed extensions for efficient NES (see [87]). The main improvements of xNES are summarized as follows.

Firstly, in order to guarantee positive definiteness, the covariance matrix $\boldsymbol{\Sigma}$ is represented as the exponential map

$$\boldsymbol{\Sigma} = \sum_{n=0}^{\infty} \frac{\mathbf{M}^n}{n!}$$

Moreover, in order to estimate the natural gradient in a computationally efficient way, the parameters are expressed in a new coordinate system where the search distribution is $\mathcal{N}(0, \mathbf{I})$. Parameter updates are then performed in this local natural coordinate system where the fisher information matrix equals the identity matrix. In each generation, natural gradients with respect to the parameters are computed in the local natural coordinates $(\boldsymbol{\delta}, \mathbf{M})$ (respecting the notation of the xNES paper). Thereafter, the updated parameters in the global coordinates (see [87] or [49]) are

$$(\boldsymbol{\delta}, \mathbf{M}) \longmapsto \left(\boldsymbol{\mu}^{'}, \mathbf{A}^{'}\right) = \left(\boldsymbol{\mu} + \mathbf{A}\boldsymbol{\delta}, \mathbf{A} \exp\left\{\frac{1}{2}\mathbf{M}\right\}\right) \tag{2.20}$$

where $\boldsymbol{\Sigma} = \mathbf{A}\mathbf{A}^{\top}$ is the Cholesky decomposition of the covariance matrix.

Algorithm 2: Exponential Natural Evolution Strategy

input: f, $\boldsymbol{\mu}_{init}$, σ_{init}, $\mathbf{B}_{init}$
output: $\boldsymbol{\mu}_G$, σ_G, $\mathbf{B}_G$
for $g = 0, \ldots, G$ **do**
  for $i = 1, \ldots, \lambda$ **do**
    $\mathbf{s}_i \sim \mathcal{N}(0, \mathbf{I})$
    $\mathscr{G}_i \leftarrow \boldsymbol{\mu}_g + \sigma_g \, \mathbf{B}_g \, \mathbf{s}_i$
    Evaluate genotypes and compute fitness $f(\mathscr{G}_i)$;
  end
  Sort genotypes $\{\mathscr{G}_i\}_{i=1}^{\lambda} \mid f(\mathscr{G}_i) \geq f(\mathscr{G}_j), \; \forall j \in \{i, \ldots, \lambda\}$

$$u(\mathscr{G}_i) \leftarrow \frac{\max\{0, \log(\frac{\lambda}{2} + 1) - \log(i))\}}{\sum_{j=1}^{\lambda} \max\{0, \log(\frac{\lambda}{2} + 1) - \log(j))\}} - \frac{1}{\lambda}$$

  $\nabla_{\boldsymbol{\delta}} J = \sum_{i=1}^{\lambda} u(\mathscr{G}_i) \, \mathbf{s}_i$
  $\nabla_{\mathbf{M}} J = \sum_{i=1}^{\lambda} u(\mathscr{G}_i) \, (\mathbf{s}_i \mathbf{s}_i^{\top} - \mathbf{I})$
  $\nabla_{\sigma} J = \dfrac{\mathrm{tr}(\nabla_{\mathbf{M}} J)}{d}$
  $\nabla_{\mathbf{B}} J = \nabla_{\mathbf{M}} J - \nabla_{\sigma} J \, \mathbf{I}$
  $\boldsymbol{\mu}_{g+1} \leftarrow \boldsymbol{\mu}_g + \eta_{\boldsymbol{\delta}} \, \sigma_g \, \mathbf{B}_g \, \nabla_{\boldsymbol{\delta}} J$
  $\sigma_{g+1} \leftarrow \sigma_g \exp\left\{ \dfrac{\eta_{\sigma}}{2} \nabla_{\sigma} J \right\}$
  $\mathbf{B}_{g+1} \leftarrow \mathbf{B}_g \exp\left\{ \dfrac{\eta_{\mathbf{B}}}{2} \nabla_{\mathbf{B}} J \right\}$
end

The xNES algorithm is exposed in Algorithm 2, where, for each generation g, the following main steps are accomplished:

1. λ genotypes are sampled from $\mathcal{N}(\boldsymbol{\mu}_g, \sigma_g^2 \mathbf{B}_g \mathbf{B}_g^{\top})$ by firstly sampling $\mathbf{s}_i$ from $\mathcal{N}(0, \mathbf{I})$ for all $i \in \{1, \ldots, \lambda\}$ and then transforming it to the final chromosomes as $\mathscr{G}_i = \boldsymbol{\mu}_g + \sigma_g \mathbf{B}_g \mathbf{s}_i$. Note that $\mathbf{A}_g$ is again decomposed into $\sigma_g \mathbf{B}_g$, representing the step size of the 'mutations' and the normalization of $\mathbf{A}_g$ with determinant equal to 1, respectively.
2. The sampled genotypes are evaluated into the task, swarm robotics simulated problems in our context, and a fitness value is associated to each genotype. The genotypes are ranked by their fitness values, resulting in a utility value $u(\mathscr{G}_i)$ of each individual. This procedure, known as fitness shaping, computes the utilities as exposed in the algorithm (see also [49]).
3. Natural gradients of the expected fitness with respect to the parameters in the local natural coordinate system ($\nabla_{\boldsymbol{\delta}} J$ and $\nabla_{\mathbf{M}} J$) are computed for $(\boldsymbol{\delta}, \mathbf{M}) = (0, 0)$. $\nabla_{\sigma} J$ and $\nabla_{\mathbf{B}} J$ are also obtained given $\nabla_{\boldsymbol{\delta}} J$ and $\nabla_{\mathbf{M}} J$. In the computation of $\nabla_{\sigma} J$, tr is the trace of a matrix and d is the genotype length or search space dimension.
4. Search distribution parameters are updated using NGA (see Eq. 2.19) and the mapping 2.20. Note that $\exp\{\frac{\eta_{\mathbf{B}}}{2} \nabla_{\mathbf{B}} J\}$ is the matrix exponential.

Building on top of xNES, Separable Natural Evolution Strategy (SNES), see [50], particularizes the algorithm to multivariate normal distributions with diagonal covariance matrix $\boldsymbol{\Sigma} = \boldsymbol{\sigma} \, \mathbf{I}$, with $\boldsymbol{\sigma} \in \mathbb{R}^d_{>0}$. This simplification is utterly suitable when dealing with large dimensional problems such as in neuroevolution. On the one hand, the use of the simplified covariance matrix drastically reduces the number of parameters to be optimized and, thus, requires much less fitness evaluations and generations. Moreover, it alleviates the computational cost of parameter updates. Therefore, SNES is the algorithm from the family of NES used in this Master Thesis due to its suitability in evolving ANN models.

Algorithm 3: Separable Natural Evolution Strategy

input: f, $\boldsymbol{\mu}_{init}$, $\boldsymbol{\sigma}_{init}$
output: $\boldsymbol{\mu}_G$, $\boldsymbol{\sigma}_G$
for $g = 0, \ldots, G$ **do**
  for $i = 1, \ldots, \lambda$ **do**
    $\mathbf{s}_i \sim \mathcal{N}(0, \mathbf{I})$
    $\mathscr{G}_i \leftarrow \boldsymbol{\mu}_g + \boldsymbol{\sigma}_g \odot \mathbf{s}_i$
    Evaluate genotypes and compute fitness $f(\mathscr{G}_i)$;
  end
  Sort genotypes $\{\mathscr{G}_i\}_{i=1}^{\lambda} \mid f(\mathscr{G}_i) \geq f(\mathscr{G}_j),\ \forall j \in \{i, \ldots, \lambda\}$

$$u(\mathscr{G}_i) \leftarrow \frac{\max\{0,\ \log(\frac{\lambda}{2} + 1) - \log(i))\}}{\sum_{j=1}^{\lambda} \max\{0,\ \log(\frac{\lambda}{2} + 1) - \log(j))\}} - \frac{1}{\lambda}$$

  $\nabla_{\boldsymbol{\mu}} J = \sum_{i=1}^{\lambda} u(\mathscr{G}_i)\, \mathbf{s}_i$
  $\nabla_{\boldsymbol{\sigma}} J = \sum_{i=1}^{\lambda} u(\mathscr{G}_i)\, (\mathbf{s}_i^2 - \mathbf{I})$
  $\boldsymbol{\mu}_{g+1} \leftarrow \boldsymbol{\mu}_g + \eta_{\boldsymbol{\mu}}\, \boldsymbol{\sigma}_g\, \nabla_{\boldsymbol{\mu}} J$
  $\boldsymbol{\sigma}_{g+1} \leftarrow \boldsymbol{\sigma}_g \exp\left\{\frac{\eta_{\boldsymbol{\sigma}}}{2}\, \nabla_{\boldsymbol{\sigma}} J\right\}$
end

The description of SNES will be remarkably short as it mostly uses the concepts and tools already explained in xNES. The pseudocode of SNES is gathered in Algorithm 3. It can be observed that only two parameters, namely the distribution center $\boldsymbol{\mu}$ and the diagonal of the covariance matrix $\boldsymbol{\sigma}$, are optimized. The natural gradient computations and parameter updates result from the ones in Algorithm 2 for diagonal $\boldsymbol{\Sigma}$.

Chapter 3

Materials and Methods

This chapter describes the mathematical formulation of the robotics simulator designed and implemented in this Master Thesis. It is initialized by formally defining the environment where the experiments are carried out. Subsequently, the simulated mobile robots are exposed, highlighting the differential drive system responsible of agent motion, the sensor models used to acquire insight of the environment and the actuators used to interact with other robots and with the environment. Moreover, in this chapter, the minimal communication system that robots can use to cooperate is fully described. The explanation is partitioned into the transmission and reception sides of the communication. Finally, the neural controller and evolution experimental setups are displayed from a general perspective. It specifies the hyperparameters that are common to all the experiments.

3.1 The environment

The environment where the experiments will take place is modelled as a mathematical torus. Let us define the rectangle $\mathrm{R} \subset \mathbb{R}^2$ of sizes W and H. For simplicity, and without loss of generality, $\mathrm{R} := [0, W] \times [0, H]$. Therefore, the functions f_W and f_H are defined as:

$$\begin{array}{rccc} f_W: & \mathrm{R} \times \mathrm{R} & \longrightarrow & \mathbb{R}^+ \cup \{0\} \\ & ((a_1, a_2), (b_1, b_2)) & \longmapsto & \min\{W - |a_1 - b_1|, |a_1 - b_1|\} \end{array}$$

$$\begin{array}{rccc} f_H: & \mathrm{R} \times \mathrm{R} & \longrightarrow & \mathbb{R}^+ \cup \{0\} \\ & ((a_1, a_2), (b_1, b_2)) & \longmapsto & \min\{H - |a_2 - b_2|, |a_2 - b_2|\} \end{array}$$

Once the torus is established, a binary relation $\mathscr{R}$, namely for $A, B \in \mathrm{R}$, is defined as

$$A \mathscr{R} B \iff f_W(A, B) = f_H(A, B) = 0.$$

In this situation, a quotient set $\mathcal{T}$ is defined as $\mathcal{T} := \mathrm{R}/\mathscr{R}$, which is referred as torus hereafter. The equivalence classes in this quotient set are defined as follows. Let $A = (a, b) \in \mathrm{R}$ and its equivalence class, denoted by $[A]$, be

$$[A] = \begin{cases} \{A\} & \text{if } A \in (0, W) \times (0, H) \\ \{(a, 0), (a, H)\} & \text{if } A \in (0, W) \times \{0, H\} \\ \{(0, b), (W, b)\} & \text{if } A \in \{0, W\} \times (0, H) \\ \{(0, 0), (0, H), (W, 0), (W, H)\} & \text{if } A \in \{0, W\} \times \{0, H\} \end{cases}$$

Note that $\mathcal{T}$ is a representation of the torus obtained when folding the rectangle R (see Fig. 3.1). Intuitively speaking,

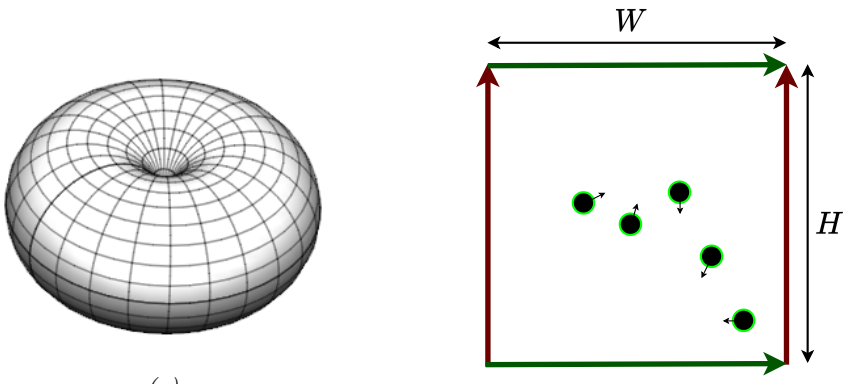


Figure 3.1: (a) 3D torus. (b) 2D flat torus, upper and lower sides and right and left sides are respectively connected.

1. the points in the interior of R are not associated with other points
2. each point on a vertical side of R is associated with the corresponding point, at the same height, on the opposite vertical side of R.
3. each point on an horizontal side of R is associated with the corresponding point, at the same distance of the y axis, on the opposite horizontal side of R.
4. the four vertices of R are associated.

In the following, taking into account this informal description, we simplify the notation by omitting the square brackets when referring to the elements in the torus $\mathcal{T}$. Now, we introduce the distance (see e.g. page 331 in [88])

$$\begin{array}{rccc} \mathrm{d}_{\mathcal{T}}: & \mathcal{T}^2 & \longrightarrow & \mathbb{R}^+ \cup \{0\} \\ & (A,B) & \longmapsto & \sqrt{f_W(A,B)^2 + f_H(A,B)^2} \end{array}$$

Therefore, in the following, we will be working in the metric space $(\mathcal{T}, \mathrm{d}_{\mathcal{T}})$.

In addition to the notion of distance between two points in $\mathcal{T}$, a definition of the angle between a vector joining two points and a reference vector will be required. For this purpose, the vector $(f_H(A,B), f_W(A,B))$ is associated to a new vector $\mathbf{v}_{(A,B)}$ defined as,

$$\mathbf{v}_{(A,B)} = (\varepsilon_W f_W,\ \varepsilon_H f_H)^\top \tag{3.1}$$

where

$$\varepsilon_W = \begin{cases} \mathrm{sign}(a_1 - b_1), & \text{if } f_W = |a_1 - b_1| \\ -\mathrm{sign}(a_1 - b_1), & \text{if } f_W = W - |a_1 - b_1| \end{cases}$$
$$\varepsilon_H = \begin{cases} \mathrm{sign}(a_2 - b_2), & \text{if } f_H = |a_2 - b_2| \\ -\mathrm{sign}(a_2 - b_2), & \text{if } f_H = H - |a_2 - b_2| \end{cases}$$

with $A \neq B$. Thus, provided that $A, B \in \mathcal{T}$, the angle between the vector $\mathbf{v}_{(A,B)}$ and a reference vector $\mathbf{u} = (1,0)^\top$ is defined as the angle $\sphericalangle_{\mathcal{T}}$ between these vectors in the Euclidean space $\mathbb{R}^2$:

$$\sphericalangle_{\mathcal{T}}(A,B) = \arccos\left(\frac{\varepsilon_W f_W(A,B)}{\mathrm{d}_{\mathcal{T}}}\right) \tag{3.2}$$

Note that the expression to which the arccos is applied belongs to the closed interval $[-1,1]$.

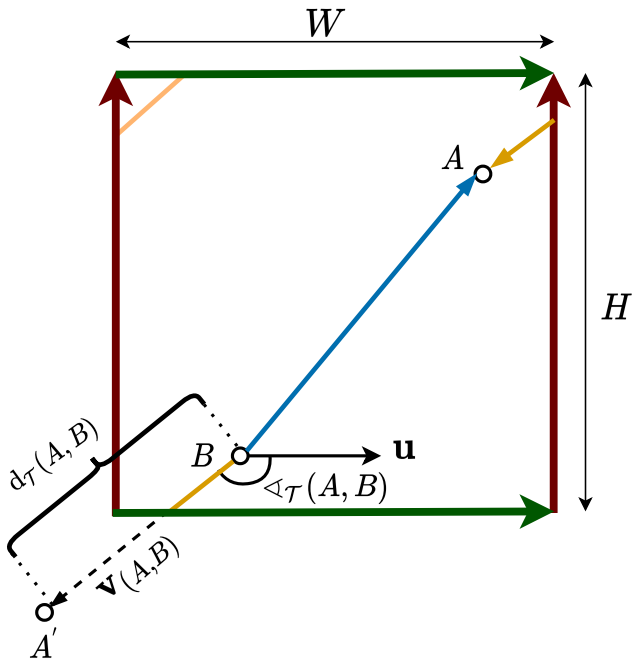


Figure 3.2: Example to clarify the distance $\mathrm{d}_{\mathcal{T}}(A,B)$ *and the angle* $\sphericalangle_{\mathcal{T}}(A,B)$ *between points* $A=(a_1,a_2)\in\mathcal{T}$ *and* $B=(b_1,b_2)\in\mathcal{T}$*. The blue vector joins* A *and* B *as it would be in* $\mathbb{R}^2$*.* $\mathbf{v}_{(A,B)}$*, in orange, is the vector joining these points considering that* $f_W = W-|a_1-b_1|$ *and* $f_H = H-|a_2-b_2|$*. The virtual point* A' *is also depicted in order to visualize more easily* $\mathrm{d}_{\mathcal{T}}(A,B)$ *and* $\sphericalangle_{\mathcal{T}}(A,B)$*.*

Fig. 3.2 depicts the idea behind the notions of $\sphericalangle_{\mathcal{T}}(A,B)$ and $\mathrm{d}_{\mathcal{T}}(A,B)$ with an example. It shows in orange color the vector $\mathbf{v}_{(A,B)}$ and in blue the vector $\overline{AB}$ as it would be in $\mathbb{R}^2$. Intuitively, the vector $\mathbf{v}_{(A,B)}$ can be more clearly visualized by representing a virtual point A' as in the figure and plotting the vector $\overline{A'B}$. However, notice that this is just a representation for understanding since it is not formally correct, as $A'\notin\mathcal{T}$.

In addition, the environment is composed by a set of world entities. In an experiment, the set of robots will be denoted as $\mathcal{R}$ and the set of lights as $\mathcal{L}$. Although there may be other kind of objects (for instance ground areas), all the experiments in the following chapters will uniquely involve robots and lights. Anyhow, all entities instantiated in the environment will have a time varying position belonging to $\mathcal{T}$. Thereafter, distances between objects are computed using $\mathrm{d}_{\mathcal{T}}$. Collisions, forces and other physics simulations are not implemented among robots because they are not explored in any of the experiments defined in this Master Thesis. Therefore, agents are physically treated as intangible point particles. This simplification can lead to the scenario of multiple robots with the same position. In such a case, it should be clarified that, even though their positions are the same in $\mathcal{T}$, the robots in $\mathcal{R}$ are different elements of the set. Fig. 3.3 shows a snapshot of the simulated world with 14 mobile robots, a red light, a yellow light, a blue light and a black ground area. Height and width of the arena are fixed to 10m in all the experiments. Although robots are modelled as particles, they are represented as black circles of radius 0.1m. The orientation of the robots is indicated using a black bar salient from the robot body.

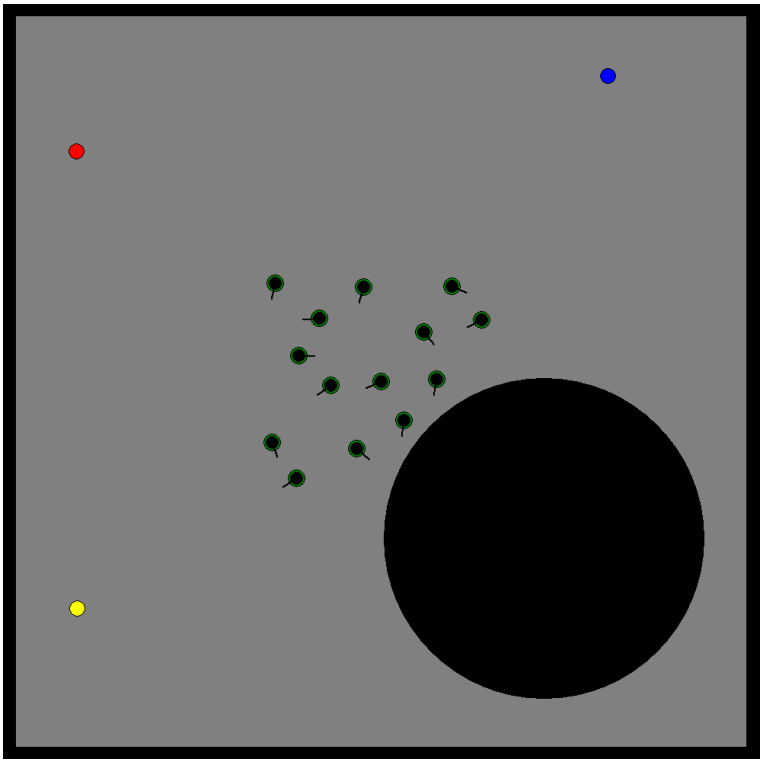


*Figure 3.3: Snapshot of the world using the developed simulator. As an example of the simulator capabilities, the environment is formed by 14 robots (in green), a black ground area and three lights of different colors (red, blue and yellow). Height and width of the arena are 10*m *each, and robots, besides being simulated as particles, are shown and black balls of radius 0.1*m.

3.2 Mobile robots

Agents are modelled as circular mobile robots with a differential drive system, a set of actuators and a set of sectorized sensors. Moreover, robots are equipped with an isotropic communication transmitter and a directional communication receiver that allow cooperation throughout communication among agents. The details of the communication will be exposed in Subsection 3.3. Within the metric space $(\mathcal{T}, \mathrm{d}_{\mathcal{T}})$ defined in the previous section, a robot $r \in \mathcal{R}$ is determined by a position $\mathbf{x}_r(k) \in \mathcal{T}$. We introduce k as the time variable of the environment. Note that while k moves in its domain, $\mathbf{x}_r(k)$ describes the trajectory of robot r within the torus $\mathcal{T}$. Moreover, a mobile robot also has a time dependent orientation $\theta_r(k)$ of its heading that will impact on its kinematics (see Subsection 3.2.1) and the receptive fields of sensors (see Subsection 3.2.2). In Fig. 3.3, the orientation $\theta_r(k)$ is represented by the small black segment attached to the robots and defining its heading. Each robot has an attached controller program that establishes the sensors and actuators to be used and that maps stimuli measured by the sensors to actions to be performed using the actuators. In all the experiments, evolved neural controllers are employed to accomplish this task (see Section 3.4).

3.2.1 Differential drive system

Robots are able to move and explore the environment by means of a differential drive system, controlling two wheels sharing a common rotation axis. Let $a_{wr}(k) \in [-\omega_{max}, \omega_{max}]$

and $a_{w\ell}(k) \in [-\omega_{max}, \omega_{max}]$ represent the robot actions for controlling the right and left wheels, respectively. In all the experiments, ω_{max} is fixed to 10cm/s. In both wheels, the upper limit denotes maximum rotation speed clockwise, lower limit means maximum rotation counterclockwise and $a_{w\ell} = 0$ or $a_{wr} = 0$ results in no wheel rotation. Furthermore, let ICC(k) be the Instantaneous Center of Curvature, $\rho_{\text{ICC}}(k)$ the distance from the center of mass of the robot to ICC(k) and L_w the distance between wheels along rotation axis. Finally, $\omega(k)$ is the instantaneous angular speed of the robot around ICC(k). $\rho_{\text{ICC}}(k)$, $\omega(k)$ and ICC(k) can be obtained using Eqs. (3.3), (3.4) and (3.5). These variables can be visualized in Fig. 3.4a. Notice that, when $a_{wr} = a_{w\ell}$, the computation of ρ_{ICC} is simply $a_{wr}(k)\,\Delta k$, and, in this situation, there is no rotation $\omega(k)$.

$$\rho_{\text{ICC}}(k) = \begin{cases} \dfrac{L_w(a_{wr}(k) + a_{w\ell}(k))}{2\,(a_{wr}(k) - a_{w\ell}(k))} & \text{if } a_{wr}(k) \neq a_{w\ell}(k) \\ \\ a_{wr}(k)\,\Delta k & \text{if } a_{wr}(k) = a_{w\ell}(k) \end{cases} \tag{3.3}$$

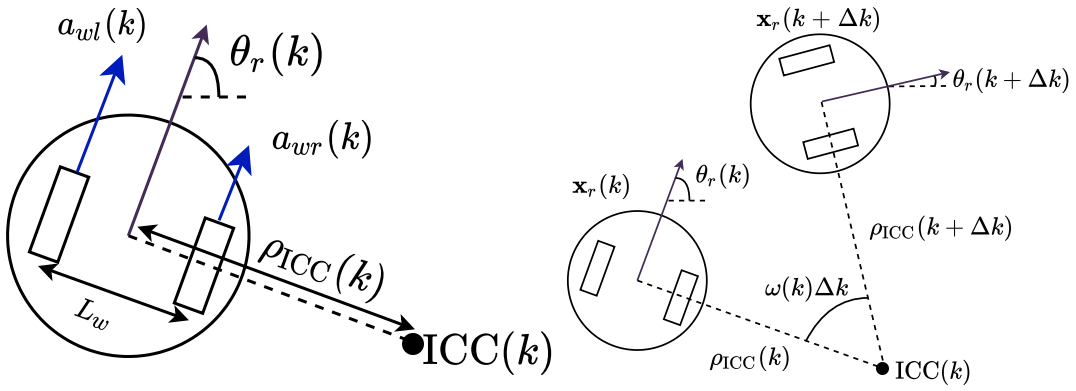
$$\omega(k) = \frac{a_{wr}(k) - a_{w\ell}(k)}{L_w} \tag{3.4}$$

$$\text{ICC}(k) = \mathbf{x}_r(k) + \rho_{\text{ICC}}(k) \begin{pmatrix} -\sin(\theta_r(k)) \\ \cos(\theta_r(k)) \end{pmatrix} \tag{3.5}$$

Finally, fixing an Euler step Δk, the position and orientation updates are stated in Eq. (3.6).

$$\left.\begin{aligned} \mathbf{x}_r(k + \Delta k) &= \begin{pmatrix} \cos(\omega(k)\,\Delta k) & -\sin(\omega(k)\,\Delta k) \\ \sin(\omega(k)\,\Delta k) & \cos(\omega(k)\,\Delta k) \end{pmatrix} (\mathbf{x}_r(k) - \text{ICC}(k)) + \text{ICC}(k) \\ \\ \theta_r(k + \Delta k) &= \theta_r(k) + \omega(k)\,\Delta k \end{aligned}\right\} \tag{3.6}$$

Fig. 3.4b shows the update of position and orientation of robots under the previously described model.



(a) Diagram of mobile robot drive system. *(b) Diagram of mobile robot kinematics step.*

Figure 3.4: Differential drive system overview.

3.2.2 Sensors

Agents are endowed with a set of sensors that can be used to partially observe the environment. The partial observability of the environment is due, mainly, to two reasons. On the one hand, robots are not able to acquire knowledge about their general positions and orientations $\mathbf{x}_r(k)$, $\theta_r(k)$ in $\mathcal{T}$. On the other hand, sensors can merely capture information within a ball of radius much smaller than H and W. This means that robots are capable of sensing the environment locally. Among the set of sensors, formed by the light sensor (LS), distance sensor (DS), ground sensor (GS) and communication receiver (RX), the agent controller is responsible of specifying the subset of sensors that should be activated in a particular experiment or problem. For instance, a single agent light follower task will only require LS information and a task that requires cooperation will employ RX sensor for being solved. Table 3.1 gathers a brief description of the currently implemented sensors. It should be mentioned that, although GS will not be used in the experiments of this Master Thesis, for completeness reasons, we have decided to include it as part of the simulator.

Sensor	Description
LS	Measures the light intensity emitted from light sources. The sensor can be sensitive to different light colors.
DS	Measures the distance to nearest neighboring robots.
GS	Measures whether the robot is placed on a colored area in the environment. GS can be sensitive to a particular ground color.

Table 3.1: Description of the available sensors.

For an experiment E, and naming $\mathcal{S} = \{LS, DS, GS, RX\}$, the subset of enabled sensors will be denoted as $\mathcal{S}_E \subseteq \mathcal{S}$. Sensors can be split into equiareal sectors so that the agent can distinguish the orientation from where a measurement is sensed. In these cases, the sensors with orientation awareness are called sectorized or directional sensors. Furthermore, for each $s \in \mathcal{S}_E$, let N_s (e.g. N_{LS} or N_{DS}) be the number of sectors of the sensor type s. Thus, the possible angles in $[0, 2\pi]$ are discretized into N_s possible measurement orientations. The quantization error of the acquired reading orientation can be reduced by increasing the number of sectors. Among the implemented sensors in the simulator, solely the ground sensor GS is not a sectorized sensor as it just detects the ground areas underneath the robot body. In the following, in order to avoid confusion, by sensor we refer to the overall sensing device encompassing the sub-sensors of all the sectors. On the contrary, a sector of a sensor will denote only the sub-sensor device pointing to the corresponding orientation of the sector. As an example, a light sensor with N_{LS} sectors would be the union of the photodiodes in all the sectors, and sector LS_j would be interpreted as the j-th photodiode.

Subsequently, let $\theta_{r,LS_j}(k)$ be the orientation of the j-th sector of LS at time instant k, which is defined as in Eq. (3.7). Notice that the orientation of sectors is relative to the robot orientation $\theta_r(k)$ and, in fact, $\theta_{r,LS_1}(k) = \theta_r(k)$ (see Fig. 3.5). Despite the particularization of Eq. 3.7 to the case of LS, the same expression can be applied to any other sectorized sensor in $\mathcal{S}$.

$$\theta_{r,LS_j}(k) = \theta_r(k) + (j-1)\,\frac{2\pi}{N_{LS}}, \quad \forall j \in \{1, \ldots, N_{LS}\} \tag{3.7}$$

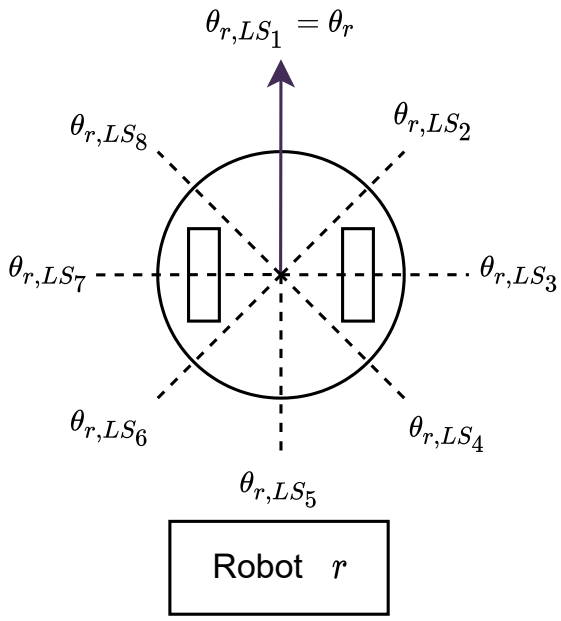


Figure 3.5: Example of a sectorized light sensor of a robot r with 8 sectors, representing the sector orientations as the dashed lines and the robot heading orientation θ_r as the arrow. Note that the orientation of the first sector equals the heading of the robot.

Apart from the communication receiver, that will be treated individually in Subsection 3.3, each sensor will result in a measurement tuple $\boldsymbol{\phi}_s(k) \in [0,1]^{N_s}, \forall s \in \mathcal{S} \setminus \{RX\}$ (for instance, $\boldsymbol{\phi}_{LS}(k)$ in the case of the light sensor) so that its components are the measured values of each sector.

The overall reading vector is denoted as $\boldsymbol{\phi}(k)$ and it is the vector concatenation of the measurements of all the enabled sensors in $\mathcal{S}_E$, as in Eq. (3.8).

$$\boldsymbol{\phi}(k) = \text{vecconcat}\left(\{\boldsymbol{\phi}_s(k),\ \forall s \in \mathcal{S}_E\}\right) \tag{3.8}$$

where, assuming some fixed ordering of the elements of $\mathcal{S}_E$, vecconcat is defined as the vector concatenation of the vectors in $\{\boldsymbol{\phi}_s(k),\ \forall s \in \mathcal{S}_E\}$. Then $\boldsymbol{\phi}(k)$ is the final tuple that will be provided to the controller to generate consequent actions.

The computation of the reading $\boldsymbol{\phi}_s(k)$ of each sector will be explained in the remaining part of this subsection. Firstly, $\phi_{GS}(k)$ is taken as one if a ground area is detected underneath the robot and zero otherwise. Alternatively, the readings of LS sensor are modeled using Eq. (3.9) for each sector $j \in \{1, \ldots, N_{LS}\}$.

$$\boldsymbol{\phi}_{LS,j}(k) = \sum_{\ell \in \mathcal{L}} c_\ell \left(e^{-\lambda^{\rho}_{LS}\, \rho_\ell(k) - \lambda^{\varphi}_{LS}\, \varphi_{\ell,j}(k)} + \varepsilon_{LS} \right) \tag{3.9}$$

where,

- $\rho_\ell = \mathrm{d}_{\mathcal{T}}(\mathbf{x}_r, \mathbf{x}_\ell)$ is the distance between the agent position and the position of light source ℓ.
- $\varphi_{\ell,j}(k)$ relates the angle $\sphericalangle_{\mathcal{T}}$ between the vector joining the agent position ($\mathbf{x}_r$) and the position of ℓ ($\mathbf{x}_\ell$) with the orientation of the j-th sector ($\theta_{r,LS_j}(k)$). More precisely, it is defined as,

$$\varphi_{\ell,j}(k) = \min\left\{ |\theta_{r,LS_j}(k) - \sphericalangle_{\mathcal{T}}(\mathbf{x}_r(k), \mathbf{x}_l(k))|, 2\pi - |\theta_{r,LS_j}(k) - \sphericalangle_{\mathcal{T}}(\mathbf{x}_r(k), \mathbf{x}_l(k))| \right\} \tag{3.10}$$

Essentially, it measures the misalignment of the measured light beam with respect to the sector orientation.

- λ^{ρ}_{LS} is a decaying constant to calibrate the distance attenuation.
- λ^{φ}_{LS} is a decaying constant to calibrate the misalignment attenuation. Notice that large values of λ^{φ}_{LS} produce overlapped coverage between contiguous sectors while small values may produce blind spots.
- $\varepsilon_{LS} \sim \mathcal{N}(0, \sigma^2_{LS})$ is the white noise attached to the measurement, where $\mathcal{N}$ denotes the normal distribution.
- $c_\ell \in \{0, 1\}$ is a bit stating if the received light is of the same color as the one towards which LS is sensitive ($c_\ell = 1$).

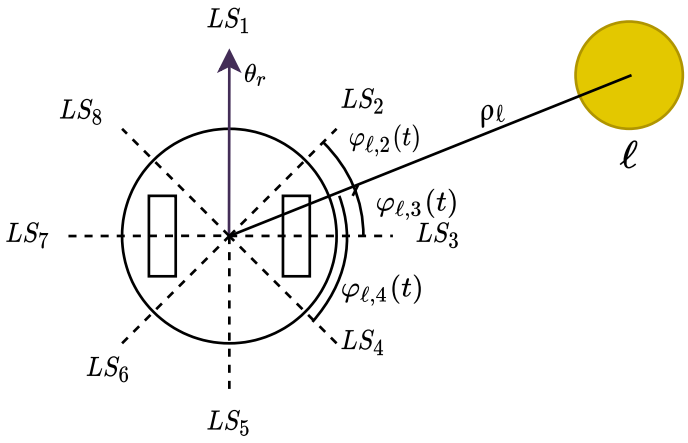


Figure 3.6: Example of a sectorized light sensor of a robot. In this case, the light sensor has 8 equispaced sectors LS_j that capture the light intensity within a coverage area. ρ_ℓ is the $\mathrm{d}_{\mathcal{T}}$ distance of the vector joining the light source position and the center of mass of the robot. Additionally, $\varphi_{\ell,j}$ is the angle between the aforementioned vector and the orientation of sensor of sector j (dashed lines).

The meaning of $\varphi_{\ell,j}(k)$ and ρ_ℓ is clearly illustrated in Fig. 3.6, where an example of a light mostly incident on sectors 2 and 3 is shown. Finally, notice that the contributions of light sources are added to the final reading. In order to bound the measurement between 0 and 1, the reading is saturated at $\boldsymbol{\phi}_{LS,j}(k) = 1$ (not shown in the formula). Fig. 3.7 displays the noiseless coverage of a sector of LS for $\lambda^{\rho}_{LS} = 0.05\mathrm{cm}^{-1}$ and $\lambda^{\varphi}_{LS} = 1\ \mathrm{rad}^{-1}$.

The simulation of DS sensor is remarkably similar to LS model. In this case, the reading of DS, is computed as in Eq. 3.11.

$$\boldsymbol{\phi}_{DS,j}(k) = \max_{r \in \mathcal{R}} \left\{ e^{-\lambda^{\rho}_{DS}\, \rho_r(k) - \lambda^{\varphi}_{DS}\, \varphi_{r,j}(k)} \right\} + \varepsilon_{DS} \tag{3.11}$$

where $\rho_r(k)$ and $\varphi_{r,j}(k)$ mean almost the same as $\rho_\ell(k)$ and $\varphi_{\ell,j}(k)$ in Eq. 3.9. More precisely, the only difference is that the former variables are computed using a robot r as

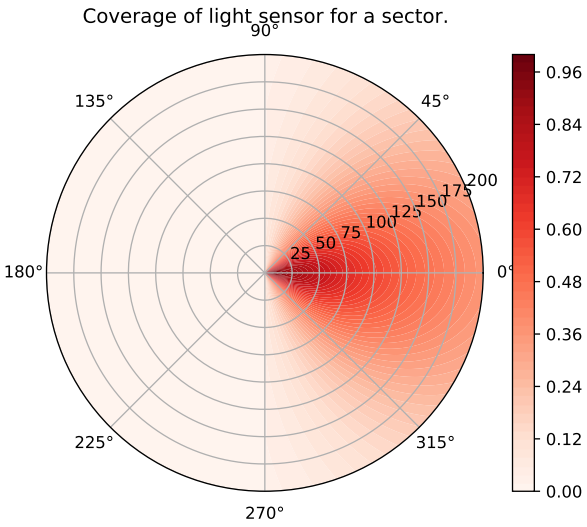


Figure 3.7: Noiseless coverage of a sector of LS for $\lambda_{LS} = 0.05\text{cm}^{-1}$. Radius is expressed in millimetres.

target and the latter ones use a light source ℓ as target object. Similarly, λ^{ρ}_{DS}, λ^{φ}_{DS} and $\varepsilon_{DS} \sim \mathcal{N}(0, \sigma^2_{DS})$ will be adjusted in order to simulate the behavior of IR sensors. The most noticeable difference between Eqs. 3.9 and 3.11 is that only the nearest robot to a sector is used to construct the reading, as opposed to LS where all contributions are added.

In order to conclude the description of the simulated sensors, Table 3.2 gathers the default parameters of each sensor. These parameters will be used in subsequent experiments unless otherwise specified.

Sensor	Range	Num. Sectors	Distance decaying constant	Misalignment decaying constant	Noise variance
LS	80cm	6	0.05cm^{-1}	$1\ \text{rad}^{-1}$	0.1
DS	60cm	4	0.05cm^{-1}	$1\ \text{rad}^{-1}$	0.1
GS	100cm	-	-	-	0

Table 3.2: Set of default parameters of the sensors. In the case of GS, the range is actually determined by the radius of the ground area and not by the sensor.

3.2.3 Actuators

Actuators provide agents with the capability of interacting with the environment or with other robots. For this purpose, the implemented actuators are wheel actuator (WA), LED actuator (LED) and wireless transmitter (TX). Analogous to the sensor set $\mathcal{S}$, the set of the mentioned actuators will be referred as $\mathcal{A} = \{\text{WA}, LED, TX\}$. Similarly, the subset of actuators enabled by a controller in an experiment E is denoted as $\mathcal{A}_E \subseteq \mathcal{A}$.

The agent controller interacts with the actuators through actions. For instance, the wheel actuator WA controls the robot wheel angular speeds, that define the robot kinematics (see

Subsection 3.2.1), by means of the action vector

$$\mathbf{a}_{\mathrm{WA}}(k) = (a_{wr}(k),\, a_{w\ell}(k))^{\top} \in [-1, 1]^2$$

Additionally, the LED actuator, is controlled with the action $a_{LED}(k) \in \{0, \ldots, C\}$, where C is the number of colors. In this situation, $a_{LED}(k) = 0$ turns off the LED and non zero values of $a_{LED}(k)$ turn on the LED photodiode with one of the C possible colors. If only one color is considered ($C = 1$), then $a_{LED}(k) \in \{0, 1\}$. LED actuators do not aim at serving as communication means. Their objective is to notify the individual solution of the agent to the collective task to be solved. An example of this statement is the task of selecting a leader of the group, in which robots can claim leadership by turning the LED on. Other robots will only be able to know that decision if the mentioned agent decides to correlate the communication transmitter action with the LED actuator state. The communication transmitter will be explained in more detail in the next section.

To conclude the actuator description, the overall action vector resulting from the vector concatenation of all enabled actuators in an experiment is shown in Equality 3.12. It is the vector concatenation $\mathbf{a}(k)$ of the actions of the previously mentioned actuators.

$$\mathbf{a}(k) = \mathrm{vecconcat}\left(\{\mathbf{a}_n(k),\ \forall n \in \mathcal{A}_E\}\right) \tag{3.12}$$

3.3 Communication techniques

This section describes the simulated communication techniques that the robots can harness in order to solve cooperative multi-agent problems. Agents can isotropically emit an M dimensional message that can be received by other robots in a low range. Receiver agents can be aware of the orientation from where the message was received, as they have a separate receiver in each sector. It should be mentioned that this directional communication is based on IR photodiodes for both transmission and reception. An agent can be either in send mode or in relay mode. Send mode means that the robot controller creates a novel message, based on the incoming message from its neighborhood, and broadcasts it to its vicinity. On the contrary, relay mode refers to broadcasting a copy of the input message. The commutation among these states will be performed by the agent controller. Specifically, provided that the robot is controlled by a CTRNN, the next state is managed by a motor neuron of the network. A maximum number of hops is fixed so that old messages eventually stop from being relayed. Moreover, messages have an attached identifier of the robot that initially generated the message content in send mode. Therefore, an agent avoids receiving echoes of its own messages by filtering out incoming packets with its own identifier.

3.3.1 Transmission

Each robot is capable of sending, isotropically, a vector message at each environment simulation cycle. The transmitted information is emitted with a simulated limited power, so that it can be received up to a fixed range with a bit error rate lower than a threshold. Let $\mathbf{a}_{TX}(k) \in [0, 1]^M$ be the communication action directly generated by the robot neural controller in response to input sensed stimuli. $\mathbf{a}_{TX}(k)$ represents the elaborated message of length M. As a postprocessing stage before emitting the message, $\mathbf{a}_{TX}(k)$ is quantized in order to be represented as one fixed cluster or symbol in a finite set. In this situation, let M^* be the number of symbols to be considered in the experiment. Then, we introduce the set $\mathcal{C}$ of possible symbols as the square lattice,

$$\mathcal{C} = \left\{0, \frac{1}{K-1}, \ldots, \frac{K-2}{K-1}, 1\right\}^M \tag{3.13}$$

where $K = \left\lceil \sqrt[M]{M^*} \right\rceil$.

For every time step k, the message quantization is defined as the following transformation,

$$\begin{array}{rccc} \mathcal{Q}: & [0,1]^M & \longrightarrow & \mathcal{C} \\ & \mathbf{a}_{TX}(k) & \longmapsto & \mathbf{m}_{TX}(k) \end{array}$$

where $\mathcal{Q}$ is fully described in Algorithm 4. The selected symbol for a raw message $\mathbf{a}_{TX}(k)$ is stochastically selected using the categorical distribution on the elements of $\mathcal{C}$ and probabilities of each element $\mathbf{p}$. Element probabilities are the result of a softmax function defined as follows. For every $\mathbf{z} \in \mathbb{R}^J$,

$$\text{softmax}(\mathbf{z})_i = \frac{e^{\mathbf{z}_i}}{\sum_{j=1}^{J} e^{\mathbf{z}_j}},\ \forall i \in \{1, \ldots, J\} \tag{3.14}$$

Algorithm 4: Message Quantization $\mathcal{Q}$

input: $\mathbf{a}_{TX}(t)$, $\mathcal{C}$, β
output: $\mathbf{m}_{TX}(t)$
$\mathbf{d}_i = \|\mathbf{a}_{TX}(k) - \mathbf{c}_i\|_2\,,\ \ \forall i \in \{1, \ldots, \#(\mathcal{C})\}$
$\mathbf{p} = \text{softmax}\left(\beta \cdot \left(1 - \frac{\mathbf{d}}{\sqrt{M}}\right)\right)$
$\mathbf{m}_{TX}(k) \sim \text{Cat}(\mathcal{C}, \mathbf{p})$

In this case, let β be a constant that controls the balance between the element probabilities. Then the components of the vector

$$\mathbf{z} = \beta \cdot (1 - \mathbf{d}/\sqrt{M}) \in [0,1]^{\#(\mathcal{C})},$$

provided as input of the softmax transformation, will be inversely proportional to the Euclidean distances $\mathbf{d}$ to the symbols in $\mathcal{C}$. $\#(\mathcal{C})$ denotes the cardinality of the set $\mathcal{C}$. Thereafter, the nearest symbols to $\mathbf{a}_{TX}$ will have largest probabilities to be selected. β is a constant that controls the balance among the element probabilities. Under the frame of the currently described communication, β can modulate the noise of the transmission medium. If $\beta \to 0$, the symbol probabilities will tend to $1/\#(\mathcal{C})$, resulting in a purely noisy communication. In contrast, provided that $\beta \to \infty$, the probabilities will tend to be sparse and the chosen symbol will approximate to the one whose Euclidean distance to the raw message is minimum:

$$\underset{\mathbf{c} \in \mathcal{C}}{\text{argmin}}\{\|\mathbf{a}_{TX}(k) - \mathbf{c}\|_2\}$$

Therefore, the aim of using the softmax function as cluster probabilities instead of a deterministic selection of the nearest symbol is twofold. Firstly, it allows a more realistic transmission simulation by tuning β and producing classification errors among near symbols. Additionally, when message generation is evolved or learnt, it provides a mechanism for promoting the exploration of new coding schemes, trying to avoid local optima.

3.3.2 Reception

As described in the previous subsection, agents are able to emit an omnidirectional message $\mathbf{m}^r_{TX}(k) \in \mathcal{C}$ (notice that superindex r is introduced in order to emphasize that the message has been emitted by the robot r). Messages can be received by other neighboring robots to the sender that capture incoming messages by means of a sectorized communication sensor. Thereafter, an agent can be aware of the sector where the sender is positioned. Thus, the robot perimeter is split into N_{RX} equiareal sectors, so that each of them will perceive only the message of the nearest neighbor within the corresponding sector coverage. Now, we formally introduce the definition of the message received in sector j. Let $\mathcal{R}_j$ be the subset of $\mathcal{R}$

containing the robots in the coverage area of sector j. Furthermore, if $\mathcal{R}_j \neq \emptyset$, we define r_j as the robot in $\mathcal{R}_j$ whose sent message's normalized signal intensity is maximum (see Eq. 3.11). Then, the received message is defined as

$$\mathbf{m}_{RX,j} = \begin{cases} \mathbf{m}_{TX}^{r_j} & \text{if } \mathcal{R}_j \neq \emptyset \\ (\underbrace{0,\ldots,0}_{M \text{ times}})^T & \text{if } \mathcal{R}_j = \emptyset \end{cases}$$

Among the received messages from all sectors, only one of them is finally selected as sensor measurement. This restriction is established with the aim of mimicking, as far as possible, real swarm robotics communications with technologies such as IR. Therefore, some selection mechanism has to be fixed. The selection scheme consists in randomly choosing an input message from those whose sender agent identifier differs from the receiving robot identifier. This utterly avoids sensing echoed messages previously emitted (recall that a robot can relay messages). If no incoming messages are sensed, then a zero vector is received.

The outcome of the previously described message selection scheme will be the sector $j^{'}$ whose message $\mathbf{m}_{RX,j'}(k)$ was picked. For simplicity, the selected message is just denoted as $\mathbf{m}_{RX}(k) = \mathbf{m}_{RX,j'}(k)$. It was already mentioned in Chapter 1 that the aim is to design a set communication mechanics that allow the emergence of both abstract and situated communication, depending on the optimization. Therefore, the received message is accompanied by a set of communication variables that contextualize the message content within the environment. These context variables are the signal strength, the orientation of the sector from where the message was received (θ_{RX}), the orientation of the sector of the sender robot $r_{j'}$ from where the message was transmitted (θ_{TX}) and the communication state or mode, that can be either relay or send. Firstly, owing to the fact that the communication simulates IR technology, the signal strength $\boldsymbol{\phi}_{DS}$ of the received message is computed using the distance sensor reading for sector $j^{'}$ as exposed in Eq. 3.11. Additionally, θ_{RX} and θ_{TX} correspond to the orientations of the corresponding sector according to Eq. 3.7, using the receiver and transmitter robots with the corresponding sensing and sending sectors. Both angles are encoded as the unit 2D vectors:

$$\text{Reception orientation} \longrightarrow (\cos(\theta_{RX}(k)),\ \sin(\theta_{RX}(k)))^\top$$

$$\text{Transmission orientation} \longrightarrow (\cos(\theta_{TX}(k)),\ \sin(\theta_{TX}(k)))^\top$$

Equality 3.15 shows the overall reading of the communication receiver $\boldsymbol{\phi}_{RX}$ that is provided to the neural controller. Notice that it is the concatenation of the message content and the context variables.

$$\boldsymbol{\phi}_{RX}(k) = \begin{pmatrix} \mathbf{m}_{RX}(k) \\ \boldsymbol{\phi}_{DS}(k) \\ \cos(\theta_{RX}(k)) \\ \sin(\theta_{RX}(k)) \\ \cos(\theta_{TX}(k)) \\ \sin(\theta_{TX}(k)) \\ MODE \end{pmatrix} \tag{3.15}$$

It should be clarified that each experiment uses different context information depending on the task mechanics and requirements. Chapter 5 settles the precise communication variables used in each problem.

3.4 Neural controller and evolution details

3.4.1 The neural controller

A robot controller can be described as a black box that receives states measured by sensors as inputs and returns actions as outputs. In general, this state-action mapping can be any kind of transformation that defines how the agent should behave. Formally, the controller can be defined as,

$$\mathbf{a}(k) = \text{controller}\left(\boldsymbol{\phi}(k)\right) \tag{3.16}$$

Despite the fact that a robot controller can be handcrafted, such as controllers using the Reynold rules [11] for the emergence of biologically inspired bird flocking, special attention is payed to the neural controllers. A neural controller implements the mapping exposed in Eq. 3.16 with an optimized artificial neural network. Specifically, continuous-time recurrent neural networks (see Chapter 2) are used as neural controllers in this Master Thesis. The CTRNN will perform a non-linear time-dependent transformation of the input state $\boldsymbol{\phi}(k)$ to the action vector $\mathbf{a}(k)$. CTRNN model can harness its dynamical properties to generate actions not only based on current stimuli but also using recent past events. The generic neural architecture is displayed in Fig. 3.8.

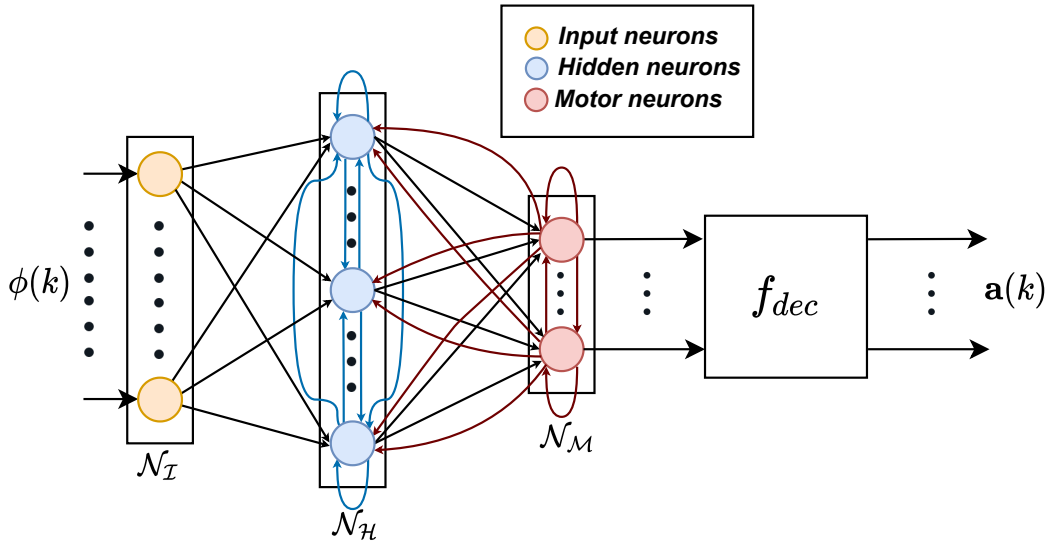


Figure 3.8: Generic architecture of the CTRNN based neural controller.

It is composed by an set of input nodes $\mathcal{N}_\mathcal{I}$ representing the stimuli vector, an ensemble of hidden neurons $\mathcal{N}_\mathcal{H}$ as main processing units and an output layer formed by motor neurons $\mathcal{N}_\mathcal{M}$. $\mathcal{N}_\mathcal{I}$ are connected to $\mathcal{N}_\mathcal{H}$, $\mathcal{N}_\mathcal{H}$ outputs feed $\mathcal{N}_\mathcal{M}$ neurons and $\mathcal{N}_\mathcal{M}$ are fed back to the hidden ensemble. Both $\mathcal{N}_\mathcal{M}$ and $\mathcal{N}_\mathcal{H}$ have self recurrent synapses. The architecture presented in Fig. 3.8 is a generic structure that will be particularized in each of the experiments presented in Chapter 5. Firstly, the most noticeable alteration between experiments is that the number of input and motor neurons will depend on the set of sensors $\mathcal{S}_E$ and actuators $\mathcal{A}_E$ enabled. Additionally, the number of hidden units H (cardinality of $\mathcal{N}_\mathcal{H}$) and the connectivity p_c of the neural network will be equally established in the experiments. p_c^{ij} defines the probability of connection between presynaptic j-th neuron and postsynaptic i-th neuron with

an existing synapse between them. If two neurons do not share a synapse in the depicted architecture, their probability of connection is zero (for instance, input neurons cannot be directly connected to motor neurons). The implementation of p_c^{ij} allows the reduction of the number of synapses accordingly to the decrease of p_c^{ij} probabilities. Clearly, the unweighted adjacency matrix of the CTRNN, defining its connections, is stochastically sampled from Bernoulli distribution $\mathcal{B}(p_c^{ij})$ $\forall i \in \{1, \ldots, N\}$ $\forall j \in \{1, \ldots, N + I\}$ only once. Thereafter a seed is fixed and the same connections are kept throughout all the trials of an experiment. Decoders or decoding functions f_{dec} transform the motor neuron activities into actions. Wheel actions and wireless transmitter actions will basically use an identity decoder while LED action and communication state action (relay or send) are sampled stochastically using the neuron activities as probabilities (using Bernoulli on the sigmoid outputs if action is boolean or softmax if there is more than one LED color).

The simulator experiments operate at two different time scales. k is the time variable of the environment, so that, at each time instant of k, the agents read the current partially observable state using their sensors and perform actions. On the contrary, t is the time variable at the neuronal dynamics time scale. Taking into account the concept of attractors in dynamical systems (see Chapter 2) and considering that the random selection of input messages produces a high frequency stimuli signal, it is reasonable to set the neuronal dynamics at a much faster time scale than the world dynamics. In general, the relation between time scales is:

$$t = \delta\, k,\ \delta > 1$$

This condition permits the membrane voltages $\mathbf{v}(t)$ of the CTRNN reaching the attractor, if any, corresponding to the current input stimuli vector. The action at each world time step of k is decoded using the activities of the motor neurons at instants $t = \delta k$ using f_{dec}. Similarly, the input stimuli is maintained constant in between sampling instants of the environment:

$$\boldsymbol{\phi}(t+1) = \boldsymbol{\phi}(t),\ \forall t \in [\delta\, k,\, \delta\,(k+1) - 1),\ \forall k \in [0,\, \infty)$$

In all the experiments, δ is set to 20 so that during each environment step, the CTRNN dynamics are updated 20 times. Considering a real hardware implementation of the controller, the δ steps of the CTRNN can be scheduled in between sensor and actuator executions when the microcontroller would be idle otherwise.

3.4.2 The genotype and phenotype

The parameters of the CTRNN to be evolved are the synapse strengths in $\mathbf{W}$, the neuron time constants $\boldsymbol{\tau}$, the neuron biases $\boldsymbol{\beta}$ and the neuron gains $\mathbf{g}$. These optimization parameters are merged into a single vector to generate a genotype $\mathscr{G}$ stored as an individual in the evolved population. In contrast, the phenotype is the complete CTRNN. Before defining the phenotype to genotype transformation and its inverse, we consider the distinction between unconnected neurons and synapses whose evolved strengths tend to zero. Let $\mathbf{M} \in \{0,1\}^{N\times(N+I)}$ be the fixed unweighted adjacency matrix that describes the CTRNN topology. It is not evolved and its components are sampled as $\mathbf{M}_{ij} \sim \mathcal{B}(p_c^{ij})$, where p_c^{ij} is the above mentioned connection probability between neurons. The non-zero elements of the element-wise multiplication $\mathbf{M} \odot \mathbf{W}$ indicate the weights to be evolved. More precisely, the vector formed by all the evolvable weights $\boldsymbol{\vartheta}$ is,

$$\boldsymbol{\vartheta} = \text{vectorize}\left(\{\mathbf{W}_{ij} \mid \mathbf{M}_{ij} \odot \mathbf{W}_{ij} > 0\ \forall i \in \{1, \ldots, N\}\ \forall j \in \{1, \ldots, N + I\}\}\right)$$

The search space is constrained to a hypercube by defining maximum and minimum values of each optimization variable. Using these minimum and maximum bounds, $\boldsymbol{\vartheta}$, $\boldsymbol{\tau}$, $\mathbf{g}$ and $\boldsymbol{\beta}$

are normalized to the $[0, 1]$ range. The mentioned normalization is as follows:

$$\hat{\boldsymbol{\vartheta}} = \frac{\boldsymbol{\vartheta} - w_{min}}{w_{max} - w_{min}}, \quad \hat{\mathbf{g}} = \frac{\mathbf{g} - g_{min}}{g_{max} - g_{min}}, \quad \hat{\boldsymbol{\beta}} = \frac{\boldsymbol{\beta} - \beta_{min}}{\beta_{max} - \beta_{min}}, \quad \hat{\boldsymbol{\tau}} = \frac{\log(0.5 \cdot \boldsymbol{\tau}) - \tau_{min}}{\tau_{max} - \tau_{min}}$$

Notice that the search space of $\boldsymbol{\tau}$ is logarithmic as membrane time constants can range from few milliseconds to hundreds of seconds. The final genotype is the vector concatenation exposed in Equality 3.17.

$$\mathscr{G} = \begin{pmatrix} \hat{\boldsymbol{\vartheta}} \\ \hat{\mathbf{g}} \\ \hat{\boldsymbol{\tau}} \\ \hat{\boldsymbol{\beta}} \end{pmatrix} \tag{3.17}$$

When using a genetic algorithm, genotypes in a population are sampled from $\mathcal{U}(0, 1)$ (initialization is performed in the genotype space). Alternatively, when a natural evolution strategy is used, the initial mean parameter $\boldsymbol{\mu}_{init}$ of the NES is fixed to center of the search space $(0.5, \ldots, 0.5)^T$ and $\boldsymbol{\sigma}$ is initialized as $\boldsymbol{\sigma}_{init} = (0.2, \ldots, 0.2)^\top$. The population genotypes are generated from $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma})$.

Once the phenotype to genotype transformation is defined, its inverse has to be determined. In particular, the genotype vector $\mathscr{G}$ is denormalized as shown in the following set of equations:

$$\begin{aligned} \boldsymbol{\vartheta} &= \hat{\boldsymbol{\vartheta}} \cdot (w_{max} - w_{min}) + w_{min}, & \mathbf{g} &= \hat{\mathbf{g}} \cdot (g_{max} - g_{min}) + g_{min} \\ \boldsymbol{\beta} &= \hat{\boldsymbol{\beta}} \cdot (\beta_{max} - \beta_{min}) + \beta_{min}, & \boldsymbol{\tau} &= 2 \cdot 10^{\hat{\boldsymbol{\tau}} \cdot (\tau_{max} - \tau_{min}) + \tau_{min}} \end{aligned}$$

where $\hat{\boldsymbol{\vartheta}}$, $\hat{\mathbf{g}}$, $\hat{\boldsymbol{\tau}}$ and $\hat{\boldsymbol{\beta}}$ are extracted from $\mathscr{G}$. These denormalized vectors are converted into a CTRNN to create the phenotype. It should be mentioned that the same phenotype is used as neural controller for all robots in the experiments (homogeneity principle of swarm robotics). Finally, Table 3.3 exposes the search space bounds of each variable vector subject to evolution. The $[-1, 0.75]$ range of $\boldsymbol{\tau}$ is mapped to $[2 \cdot 10^{-1}, 2 \cdot 10^{0.75}] = [0.2, 11.25]$ seconds. Equivalently, the time constant search space is more suitably represented in terms of the Euler step of the CTRNN as $[2\Delta t, 112.5\Delta t]$. This search space of time constants allows the existence of a variety of neuronal regimes. For example, low time constants allow the rapid reaction to abrupt changes in the input stimuli, while large time constants permit the formation of working memory at a much slower time scale. The weights are constrained to

Variable	Min.	Max.
$\boldsymbol{\vartheta}$	-3	3
$\boldsymbol{\tau}$	-1	0.75
$\boldsymbol{\beta}$	-1.5	1.5
$\mathbf{g}$	0.05	5

Table 3.3: Search space constrains of each of the denormalized optimization variables.

the interval $[-3, 3]$, because we heuristically found this range to be a good tradeoff between strong amplification and saturation avoidance. Note that synapse weights are both positive and negative because inhibition is as important as excitation in dynamical neural networks for the generation of complex dynamics. Similarly the search space of biases is designed according to the same argument. Finally, the gain values are restricted uniquely to positive values, avoiding the scenario of zero gain. Moreover, the upper bound is 5, as larger gains would produce an utterly discontinuous neuron activation (tending to a Heaviside activation).

3.4.3 Evolution hyperparameters

As already mentioned in previous chapters, the neural controller of each task or experiment will be evolved using various soft computing algorithms. Specifically, genetic algorithms and natural evolution strategies are explored. In the case of natural evolution strategies, SNES algorithm is used as a suitable evolution strategy for high dimensional optimization problems as neuroevolution. Most of the hyperparameters of these algorithms will be fixed and the same for all the experiments unless otherwise specified. Table 3.4 gathers the hyperparameters of GA and SNES (see Chapter 2 for a detailed explanation of these parameters). Fistly, the

GA		SNES	
P	100	P	100
p_{mut}	0.05	$\eta_{\boldsymbol{\mu}}$	1
p_{cx}	0.9	$\eta_{\boldsymbol{\sigma}}$	$\frac{3+\log(d)}{5\sqrt{d}}$
N_{elites}	3	$\boldsymbol{\mu}_{init}$	$(0.5,\ldots,0.5)^\top$
Selection	Tournament	$\boldsymbol{\sigma}_{init}$	$(0.2,\ldots,0.2)^\top$
Crossover	BLX-α		
Mutation	$\mathcal{N}(0,0.3)$		

Table 3.4: Fixed evolution hyperparameters for all the experiments

genetic algorithm operators are the tournament selection scheme with a tournament size of 3 individuals, a BLX-α recombination with $\alpha = 0.5$ and a gaussian mutation centered at the genes and with a variance of 0.3. In contrast, regarding SNES, the learning rates $\eta_{\boldsymbol{\mu}}$ and $\eta_{\boldsymbol{\sigma}}$ are based on commonly used configurations as proposed in [49, 50], being d is the genotype length. Considering that the constrained search space of $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ is the hypercube $[0,1]^d$ in both cases, $\boldsymbol{\mu}_{init}$ and $\boldsymbol{\sigma}_{init}$, shown in the table, are the initial solution candidates. Recall that $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ are not the genotypes but the parametrization of the random distribution used to generate genotypes. In both GA and SNES, the population size P is fixed to 100 in all the experiments.

Chapter 4

The Simulator

4.1 General overview

This chapter presents the software simulator developed in Python programming language for the simulation of swarm robotics experiments. The implemented code has been uploaded to Github [1], where it can be consulted. The simulated robots are principally controlled using recurrent neural networks, albeit other non neural controllers can be also implemented. Another pillar of the simulator is the use of evolutionary computation for the optimization of the neural controller parameters. Both artificial neural networks and evolutionary computation algorithms are implemented from zero and molded to fulfill the requirements of the simulator. Moreover, an important remark is that the simulator is mainly focused on the use of spiking neural networks and biologically plausible neuron models, albeit these kind of RNNs are not explored in this Master Thesis. Therefore, many aspects and design decisions of the software are adapted to fullfil the particular functioning of spiking neural networks. The main aim of the simulator is to serve as a resource for research and academic purposes. Consequently, the version presented in this Master Thesis can be considered as a first version of the project that will be improved in future research works. Bearing in mind the purpose of the simulator, it is important to clarify the reason behind the decision of developing a software simulator from zero. The main reason is that the simulator merges different soft computing fields (SR, evolutionary computation and ANNs) and thus, the available software suitably covers merely one of the fields. For instance, Gazebo [89], Webots [90] and ARGoS [91] are examples of popular robotics simulators, Nengo [92], Brian [93] and BindsNET [94] and examples of spiking neural network simulation libraries and DEAP [95] is a Python library oriented to evolutionary computation. The combination of these libraries and frameworks is a challenging task due to the different paradigms of functioning. Moreover, there are several functional requirements of our simulator that were not found in these packages. For instance, most of the SR simulators are based on visual interfaces and encompass many functionalities and features that are not relevant for this Master Thesis. These issues lead to a noticeable slowdown of the evolution or training stage. Another important reason is that the simulator was designed, at least for its first version, as a fast solution to evolution in SR, restricted to the sole use of Python standard libraries. For instance, all the mathematical computations are accomplished by means of the `numpy` [2] Python library, which is an utterly fast scientific computing package built on top of C language implementations. Furthermore, as it is described later on this chapter, a parallelization of the evolutionary computation algorithms is performed on the simulator, which would have been a highly complex task if other simulation libraries were used.

[1] https://github.com/r-sendra/SpikeSwarmSim
[2] https://numpy.org/

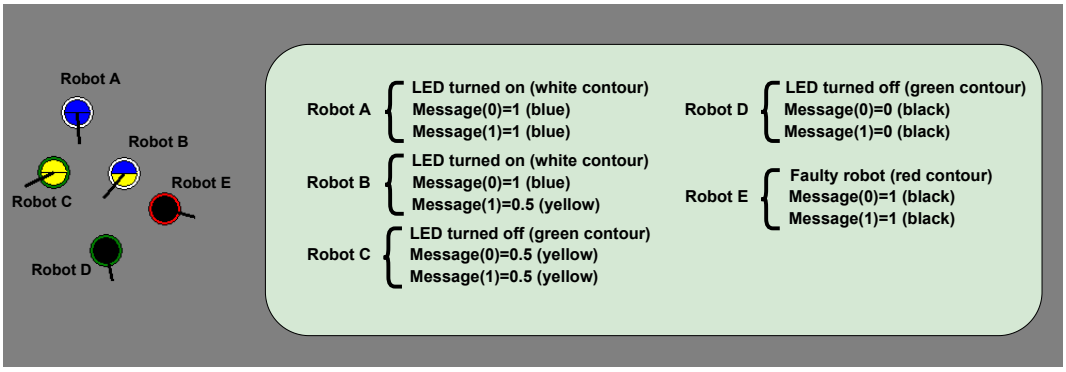


Figure 4.1: Example of the different robot colors to notify agent actions.

The simulated 2D robotics environments (described mathematically in Chapter 3), can be executed in a visual or render mode for observing the simulation at runtime. A snapshot of the simulated environment was already illustrated in Fig. 3.3. For visualization purposes, each robot notifies its actions by means of several colors. Fig. 4.2 shows a simulation snapshot with several examples of these colors. The contour of the robot indicates the status of the robot's LED. Green contour means that the LED is deactivated and white contour means that the LED is turned on (assuming only one LED color). The inner circle, divided in two semicircles, indicates the message transmitted by the robot (when the message vector has two components). The color of the upper semicircle denotes the first component of the message while the semicircle at the bottom part depicts the second component of the message. In the example, for simplification purposes, we considered that each message component has three possible values {0, 0.5, 1} that are mapped to the shown colors {black, yellow, blue}. Finally, faulty robots (see Experiment 5.1) are represented with red contours, simulating failure notification through the LED.

4.2 Simulator layers and interfaces

The simulator is designed as a stack of layers and interfaces to interconnect them and exchange relevant data. Fig. 4.2 illustrates the stack of layers and interfaces interconnecting them. The green boxes represent the different layers of the simulator while the blue boxes are the interfaces. Firstly, the different layers and the main functional role within the simulator are listed below:

- **Algorithm layer**: the outermost layer is responsible of specifying the optimization algorithm and its update rules and equations. Even though in this Master Thesis we focus on two evolutionary computation algorithms, this layer can accept any kind of optimization procedure. For instance, a future feature to be included in this layer is the implementation of algorithms from the family of reinforcement learning. The algorithm layer is also in charge of initializing, iterating and resetting the environment and its dynamics. Moreover, at the end of each simulation, the fitness function is computed in this layer. Another feature to be highlighted is that it allows executions in either learning or evaluation modes. In learning mode, the corresponding algorithm saves a checkpoint of the optimization variables and states, while in evaluation mode, diverse trial variables are gathered and stored in a dataset for a subsequent behavioral analysis.

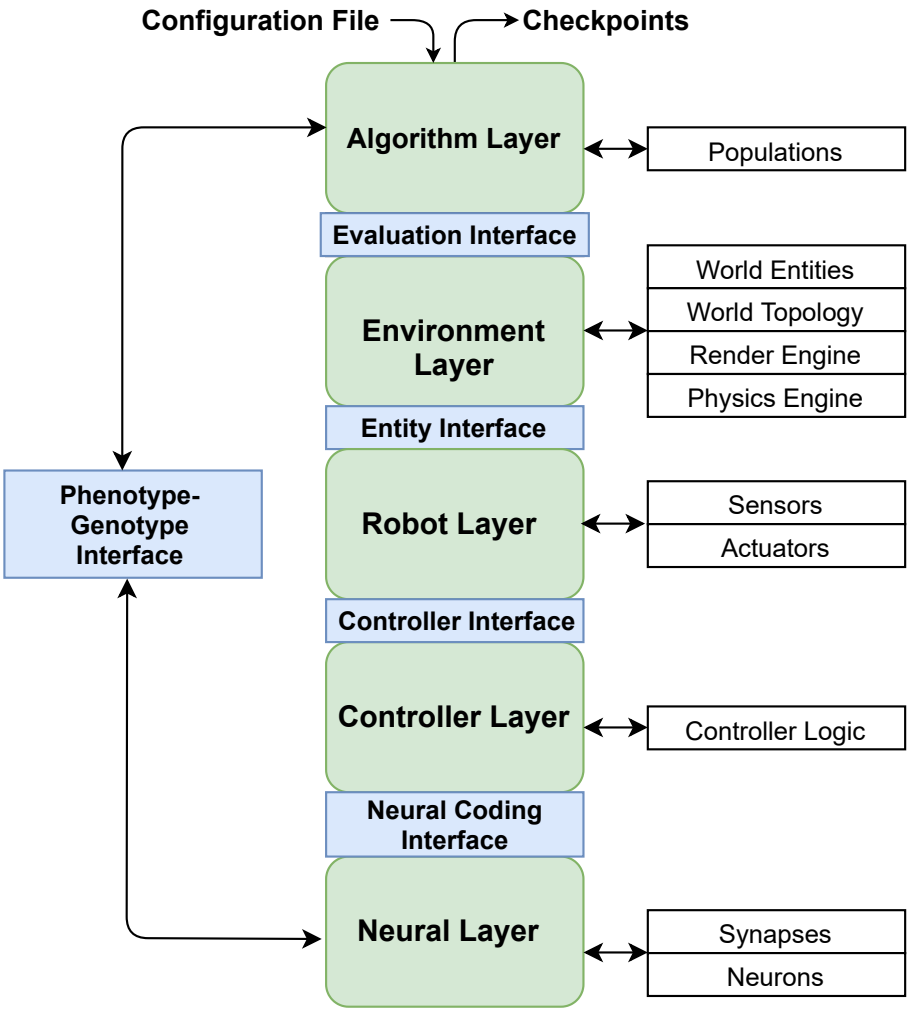


Figure 4.2: Designed simulator stack. Green boxes represent simulator layers and blue boxes indicate the interfaces.

- **Environment layer**: implements the environment dynamics and mechanics as explained in Chapter 3. It contains the instances of all the entities belonging to the environment and it is responsible of updating the dynamics of each of them when specified by the algorithm layer. In the case of the robot entities, the environment gathers the actions and states of each agent and relays it to the algorithm layer throughout the evaluation interface. Additionally, in order to update the dynamics and controllers of environment entities, information about other objects belonging to the environment has to be provided. For instance, the distance sensor of a robot has to know the positions of the other agents in its vicinity. Furthermore, robot sensors have to iterate through all the supplied entities in the environment in each simulation cycle, resulting in an inefficient implementation. Thereafter, in order to simplify the complexity of the updates of the robots' sensing process, an undirected graph is built with environment entities as nodes. Within this graph, two robots are connected if their distance is less than the IR range. Similarly, a light source is connected to a robot is their distance is less than the light coverage range. Clearly, this edge conditions are adjusted to the currently implemented entities and sensors. The graph interconnecting environment entities has to be updated in each iteration. Using this graph, only the neighborhood of a robot is provided to the next layer in order to compute

the sensor readings, resulting in a much more efficient implementation. Besides, it is worth mentioning that the environment layer is also responsible of the rendering and physics simulations. However, due to the early stage of the simulator, a very basic implementation of these features is provided in this version of the software. As a future work in subsequent versions, these functionalities have been designed as a query process to physics and render engines.

- **Robot layer**: implements the functioning of robot entities within the environment that instantiates them. Robots have access to a set of sensors and actuators that allow the insight acquisition and interaction with the environment. See Chapter 3 for a detailed explanation of sensors and actuators from a mathematical perspective. In each simulation cycle, the main steps of this layer are: (i) the reading of sensor measurements using information from the robot's neighborhood, (ii) the mapping of sensor readings to actions using the robot controller and (iii) the conversion of agent actions into environment interactions, by means of its actuators. As it is explained in the environment-robot interface later, some actions have to be validated by the environment layer before being accomplished by the actuator. The most noticeable example of this transaction is the collision management and control that, although it is not relevant in the experiments of this Master Thesis, it will play an important role in future projects. The step of mapping sensor measurements to actions is carried out by the next layer.
- **Controller layer**: defines the policy and behavior of robots within the environment by transforming sensor readings into robot actions. This layer by itself is remarkably simple in the sense that it only defines the controller logic and mechanics. In the situation in which the controller is not a neural controller, this layer is the innermost building block of the stack (e.g. when the controller is a FSM).
- **Neural layer**: extends the controller layer in the scenarios where actions are generated by a neural network. A diagram illustrating the functioning of this stage is shown in Fig. 4.3.

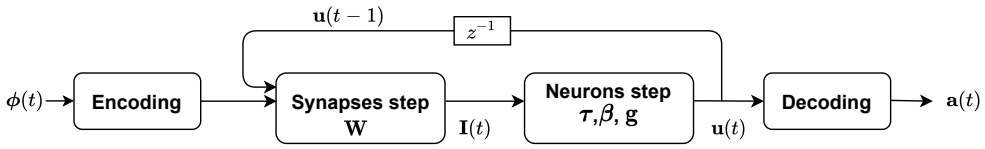


Figure 4.3: Neural layer data flow.

There, it can be observed that the process is composed by 4 main stages. The encoding and decoding phases are responsible of processing the observed state and decoding neuron firing rates into actions, respectively. These processes are considered as part of the neural coding interface (see Fig. 4.2) and, consequently, a deeper explanation is provided in the description of this interface below. Besides, it can be seen in the diagram that a RNN step is mainly composed of the sequential updates of synapses and neuron models' states. The variables within the boxes are the parameters of the synapses and neurons in the case of CTRNNs. Other types of synapse and neuron models would lead to different parameters. Moreover, even though the synapse model used in this Master Thesis is static and merely modelled as a linear transformation of the previous states and present inputs into somatic currents, there are more biologically

plausible synapse models. Specifically, synapses are governed by differential equations whose parameters state whether the connection is excitatory or inhibitory (see Chapter 7 of [67]). Furthermore, in some cases, propagation delays are also included in the synapse model for a more accurate representation. These realistic synapse models are also implemented in the simulator and used when the neural controller is based on spiking neural networks.

In addition to the described stack of layers, the simulator architecture also presents a set of interfaces that interconnect pairs of layers in order to exchange information (see Fig. 4.2). The data exchange, that is described subsequently, is related to the execution of software during the simulation phase. There is also an initialization stage in which the configuration specified through a configuration file is propagated from the uppermost algorithm layer to the neural layer in order to set up all the parameters and variables. Besides, the interfaces and their role in the simulator are the following:

- **Evaluation interface**: interconnects the algorithm and environment layers. Firstly, it sends instructions from the algorithm layer to the environment to start, reset and update the dynamics of the environment. On the other hand, the environment responses, in each simulation, cycle with the variables containing the actions and states performed by the robots. Moreover, depending on the task to be solved, the environment can also provide spatial information about the robots (e.g. their positions or their orientation). These variables are required by the algorithm in order to compute the fitness.

- **Entity interface**: Each robot belonging to the environment has an entity interface through which it receives information about neighboring entities. This information of the vicinity, which was also presented in the description of the environment layer, is used by the robots to compute the sensor readings. Subsequently, each robot transmits its state or sensor readings and the actions to be performed to the environment layer. Lastly, there is another stage in which the environment sends a validation signal of the action planned by the robot. If the validation is positive, the robot performs the action through the actuator. Fig. 4.4 shows a diagram illustrating the mentioned interaction.

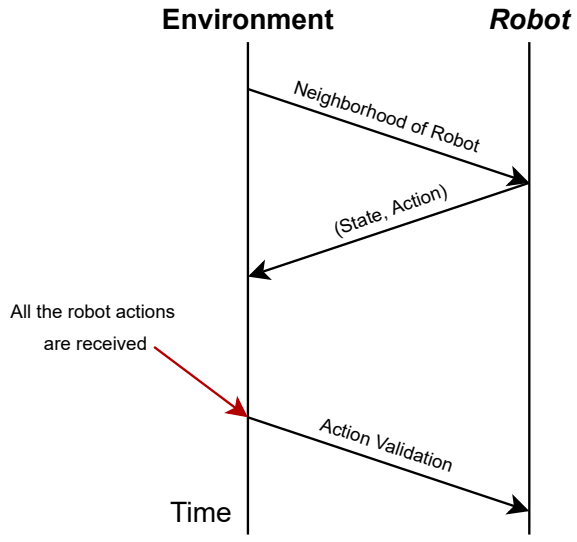


Figure 4.4: Interaction diagram between environment and robot layers.

- **Controller interface**: the controller interface transmits the measured sensor readings from the robot to the controller. Alternatively, the controller response is the set of actions resulting from the transformation of the state.
- **Neural Coding interface**: this interface performs two critical steps on the simulator. Firstly, it converts the state measured by the sensors to a data format that can be understood by the RNN. This process, known as neural encoding, is not addressed in this Master Thesis (although it is implemented in the software simulator) as it mainly involves the mapping of continuous state variables into binary spike events that can be processed by spiking neuron models. In the case of CTRNNs and firing rate neuron models, their functioning is intrinsically continuos and, thus, they do not require a state encoding. However, the neural coding interface also encompasses any sort of preprocessing of the state (e.g. dimensionality reduction or basis expansion) before being fed to the RNN. On the contrary, when the RNN step has been accomplished, this interface is also in charge of decoding the neuron firing rates (or outputs, more generally) into actions. In Chapter 3 and 5 this decoding process is mentioned as f_{dec}. The decoding can be, for instance, a Heaviside function that maps outputs in the interval $[0, 1]$ to binary actions in $\{0, 1\}$.
- **Phenotype-Genotype interface**: is an special interface as it is the only one that bypasses all the layers in order to directly connect the algorithm layer with the neural layer. As its name suggests, it is responsible of performing the bidirectional mapping between genotype vectors stored in the populations of the evolutionary algorithm to the phenotype CTRNNs in the neural layer. Note that this interface is explicitly designed for evolutionary computation algorithms. Thus, other optimization algorithms to be included in the future (e.g. RL algorithms) will require another interface.

4.3 Configuration files

All the details of the experiments in the simulator can be fully specified and gathered in JavaScript Object Notation (JSON) configuration files. This procedure implies an utterly clean, straightforward and compact manner to automatize experiments and tasks. An example of configuration file is exposed in Appendix A. There, it can be observed that the configuration file is mainly divided into `topology`, where all the RNN details are exposed, `algorithm`, for the specification of the optimization details and `world`, devoted to the declaration of the environment setup and entities to be instantiated. It is important to remark that, in the algorithm section of the configuration file, there is a JSON field called `populations` in which different evolutionary subpopulations can be specified. Even though this feature is not harnessed in the experiments of this Master Thesis, it allows the use of cooperative coevolution (see [77, 96]). In cooperative coevolution the overall genotype is segmented into different parts, so that each segment belongs to a different subpopulation being evolved independently. Subpopulations only interact when their genotype segments are combined to produce the phenotype to be evaluated. An example of usage of subpopulations in the JSON configuration file is the following:

```
1        "populations" : {
2                "p1" : {
3                    "objects" : ["synapses:weights:all"],
4                    "max_vals" : 3,
5                    "min_vals" : -3,
6                    "encoding" : "real",
7                    "selection_operator" : "tournament",
8                    "crossover_operator" : "blxalpha",
9                    "mutation_operator" : "gaussian",
```

```
                    "mating_operator" : "random",
                    "mutation_prob" : 0.05,
                    "crossover_prob" : 0.9,
                    "num_elite" : 3
                },
                "p2" : {
                    "objects" : ["neurons:bias:all", "neurons:gains:all",
                        ↪ "neurons:tau:all"],
                    "max_vals" : [1.5, 5, 0.75],
                    "min_vals" : [-1.5, 0.05, -1],
                    "encoding" : "real",
                    "selection_operator" : "tournament",
                    "crossover_operator" : "blxalpha",
                    "mutation_operator" : "gaussian",
                    "mating_operator" : "random",
                    "mutation_prob" : 0.05,
                    "crossover_prob" : 0.9,
                    "num_elite" : 3
                },
        }
```

Note that there are two subpopulations, one of them devoted to the evolution of the CTRNN weights and the other responsible of the bias, gains and time constant of neurons. Moreover, each subpopulation can be evolved using different GA operators (or optimization hyperparameters in general), albeit in the example the configurations are the same. The field `objects` of the configuration of each subpopulation conveniently leads to the next issue to be explained. Essentially, `objects` specifies the segment of the genotype to be addressed by the corresponding subpopulation. It is based on simple queries of parts of the CTRNN parameters with the shown semantics. Table 4.1 gathers all the possible queries that can be performed

Query	Description
`synapses:weights:all`	The weights of all the synapses in the RNN.
`synapses:weights:sensory`	The weights of the synapses whose presynaptic neuron is an input node.
`synapses:weights:hidden`	The weights of the synapses whose presynaptic neuron is a hidden neuron.
`synapses:weights:motor`	The weights of the synapses whose presynaptic neuron is a motor neuron.
`synapses:weights:NAME`	The weights of the synapses whose presynaptic neuron belongs to ensemble with name NAME,
`neurons:bias:all`	The biases of all the neurons.
`neurons:tau:all`	The time constants of all the neurons.
`neurons:gain:all`	The gains of all the neurons.
`neurons:decoding:all`	The decoding weights when spiking neuron models are used (not discussed in this Master Thesis).

Table 4.1: Queries to specify the set of parameters to be evolved by a subpopulation in an evolutionary computation algorithm of the simulator.

in this version of the simulator. The code implementation of the query system is sustained by two principal steps. Firstly, the synapse base class and the neuron model base class have a set of methods devoted to get, set, and initialize the pertaining parameter variable. Within these methods, there is a Python decorator that states how the query should be performed and stores the mapping between the query name and the corresponding method in a global dictionary. An example of the final query, which is not visible in the configuration file, would

be

- Get all weights of the RNN ⟹ `GET synapses:weights:all`
- Set weights to the values ⟹ `SET synapses:weights:all new_weights`
  in the array variable `new_weights`

This query system is the basis of the phenotype-genotype interface presented in previous sections.

Finally, it should be mentioned that there are some configuration options that can be directly stated when launching the simulator from the command line. For instance, the following instruction executes the simulator with the configuration of the experiment gathered in the file leader_selection_exp .json, resuming a previously saved checkpoint of the evolution and with 12 cores:

```
$ python main.py --cfg leader_selection_exp --resume --ncpu 12
```

Table 4.2 displays all the currently implemented command line arguments with their corresponding meaning. It also shows the expected argument data type and the default value.

Argument	Arg. Type	Default Value	Description
cfg	String	checkpoint	Name of the JSON configuration file to be used.
ncpu	Integer	1	Number of cores to be used.
resume	Boolean	False	Whether to resume stored checkpoint or not.
render	Boolean	True	Whether to run simulator in visual mode or console mode.
eval	Boolean	False	Whether to run simulator evaluation mode or optimization mode.

Table 4.2: Command line arguments of the simualtor.

4.4 Parallelization

Evolutionary algorithms applied to black box optimization problems generally require a large amount of time and computational cost. The bottleneck is usually the evaluation of the genotypes in the population, because it is a sequential process that commonly implies a large amount of simulation time steps and several trials to improve the reliability. Moreover, this problem is particularly critical in the case of SR, where there is a copy of the ANN phenotype per each swarm member. Thus, each single evaluation involves the concurrent execution of many ANNs.

In order to enhance the computational efficiency of the simulator and alleviate the mentioned problems, parallelization of the simulator is implemented. From a superficial perspective, we consider two levels of parallel computing in SR:

- **Population level parallelization**: is responsible of parallelizing the evolutionary algorithm so that each core evaluates a different genotype of the population in isolation to other computing units. Therefore, a copy of the environment is distributed to each

CPU core, that evaluates the corresponding genotype in it. This kind of parallelization in EAs is highly common as population sizes can be arbitrarily large and there are no shared variables or resources among different genotype evaluations. Clearly, if the population size is greater than the number of cores, the population is partitioned into mini batches that are uniformly distributed along cores. Population level parallelization is accomplished throughout CPU parallelization. The main disadvantage is that the speedup cannot be harnessed in evaluation mode.

- **Swarm level parallelization**: is responsible of assigning a different core to a different agent in the swarm in each evaluation. Thereafter, although genotypes have to be evaluated sequentially, each simulation of the environment is utterly fast due to the use of a dedicated core to each robot and, thus, to each RNN. The main issue of this kind of parallelization is that there must be a precise and accurate control of the shared resources among cores. Swarm level parallelization is specially suitable when swarm sizes are extremely large. As in the previous parallelization level, in this case, the parallel computing is also accomplished via CPU.

For this Master Thesis, the developed simulator only implements the population level parallelization because of its simplicity, lack of shared resources and remarkable computation speedup. Besides, swarm level parallelization was also discarded because swarm sizes are not very large during evolution in the proposed experiments. The parallelization is implemented by means of two alternative frameworks, namely, the multiprocessing Python library [3] and the Message Passing Interface (MPI) Python implementation [4]. Although both options are implemented and tested, multiprocessing is the one being used in the experiments of this Master Thesis. MPI is more suitable for parallelization in computer clusters.

The resulting speedup of a generation of the GA, defined as the ratio between the compute time without parallelization and the compute time with parallel computing, with 12 cores is:

$$speedup = \frac{622.15\,\mathrm{s}}{115.64\,\mathrm{s}} = 5.38$$

The speedup was computed using 10 robots, a population size of 100, 500 simulation steps and a single evaluation trial. Note that, in this situation, each core is responsible of evaluating 9 individuals (except one core that only has 1 genotype).

4.5 Future improvements

The current version of the simulator correctly covers all the requirements and needs for the experiments of this Master Thesis. However, it is thought to be used in future research projects and works or academic activities. Therefore, apart from a continuous improvement and refinement, there are several planned features to be implemented in future versions. These upgrades will be exposed in this section, following the layer order previously presented.

Firstly, an important improvement of the simulator concerns the addition of a wider variety of optimization algorithms, both evolutionary and from other artificial intelligence fields. For instance, reinforcement learning algorithms are an important improvement of the simulator and will be considered in the future for the assessment of its suitability in solving SR problems. Environment layer is the part of the simulator with most features to be refined or added. The most clear one is the incorporation of physics and collisions within the environment for a more realistic simulation. Similarly, other render options will be implemented, mainly covering the simulation of 3D graphics and more detailed robots.

[3] https://docs.python.org/3/library/multiprocessing.html
[4] https://mpi4py.readthedocs.io/en/stable/

Furthermore, as mentioned in the previous section, parallelization at the swarm level is also an interesting feature to be explored. Besides, one of the main future upgrades regarding mobile robots is the use of URDF [5] files for the accurate definition of different types of commercial robots (e.g. the epuck [6]). Regarding the controller and neural layers, the exploration of new controllers, neural models and neural coding schemes will be in continuous growth, provided that new research activities involving the simulator are carried out.

[5] http://wiki.ros.org/urdf
[6] http://www.e-puck.org

Chapter 5

Experiments

In this chapter, the different tasks or experiments considered in this Master Thesis are defined and described. Each experiment will be splitted into 6 items describing different aspects of its functioning and parameter setup. These blocks are built on top of the concepts and tools treated in Chapters 2 and 3, and are summarized as follows:

- *Task description*: each experiment starts with a description of the task functioning, mechanics and constrains.
- *Initialization*: exposes the commonly random initialization of each robot within the environment. It should be mentioned that, for a fair fitness evaluation, the same initialization seed is used for all the individuals in a generation of the GA or NES. Thus, their fitness is assessed under the same conditions. Different generations and trial evaluations have different seeds.
- *Sensors, actuators and communication*: establishes the set of sensors and actuators enabled in each experiment. Special attention is paid to their role in solving the task, the dimension and domain of its measurements or actions, among other characteristics that depend on the application. In particular, the communication receiver and transmitter will be described in more detail, specifying the dimension of the message, the number of sensing sectors and the subset of the communication context variables to be used from those defined in Eq. 3.15 (not all experiments will require the same input information).
- *Neural controller and evolution hyperparameters*: particularizes the neural controller architecture depicted in Section 3.4 to fulfill the requirements of each experiment. The number of layers, hidden neurons, connection probability, input dimension and output dimension will be exposed. Furthermore, several evolution hyperparameters, such as the evaluation time steps or the number of trials, will be set in this point. The hyperparameters that are common to all the experiments are not discussed in this chapter (see Section 3.4). It is also important to recall that all the agents in the experiment executions have a copy of the same CTRNN phenotype to be evaluated.
- *Related work*: a collection and description of related work addressing a similar experiment or task is provided, in the context of swarm robotics.
- *Fitness function*: sets and explains the fitness function of each experiment devoted to map genotypes into performance measures. The fitness function has to be handcrafted specifically for each task and it is one of the most critical design stages in GA and NES. An important consideration to be highlighted is that the fitness function will reward or penalize the robot behaviors as a group and not as sole agents. Therefore, a poor performance of one agent within the swarm penalizes the overall group performance. Finally, the fitness functions are the result of a trial average and a temporal average

of instantaneous fitness functions. By temporal average we refer to evaluating the genotypes in a task for T_E simulation time steps, computing the reward of the swarm at each instant, and computing the mean reward value along the duration of the execution. Moreover, trial average means that, instead of relying on a single sample of the fitness value, we reevaluate the genotypes N_E times in order to reduce estimation variance. The final fitness score estimate is the sample mean of the resulting fitness scores in each trial or episode. Clearly, the experiment environment and neuronal dynamics are reset and new world initialization is provided in each trial. In general, we compute the fitness as follows,

$$F = \frac{1}{N_E T_E} \sum_{n=1}^{N_E} \sum_{k=1}^{T_E} f(\mathscr{G}, k, n) \tag{5.1}$$

so that f is the actual fitness function to be designed in each experiment. n and k represent the episode and time step index variables.

It should be clarified that, in several cases, some of the mentioned building blocks will be the same in multiple experiments. In these situations, the explanation is provided only once. Thereafter, experiments with repeated behaviors or hyperparametrization will refer to the first occurrence in previous experiments.

5.1 Experiment A: Leader Selection

Task Description: The first experiment that is considered is the leader selection and preservation in a swarm of static robots. The objective is to find a cooperative behavior in which just a single agent claims the leadership of the group during the evaluation period. Cooperative solutions resulting in the swarm leader being the same during consecutive time instants will be rewarded, trying to avoid unnecessary switching of the leadership. However, it is highly desirable to find a swarm behavior capable of reacting to leader fault or loss by selecting a substitute. In order to guide evolution towards this behavioral feature, a robot becomes faulty if it has been the unique leader for 50 consecutive simulation cycles. In this context, a faulty robot means that the agent can only relay input messages (it is always in relay mode) and it cannot claim leadership. As communication is still operative after failure, the swarm graph cannot be broken or disconnected due to leader failure. Moreover, in order to avoid convergence to a solution in which the leader is constantly commuted, and robot failure is avoided as a consequence, the fitness function rewards the group proportionally to the consecutive time steps with the same leader (see fitness function description below). The size of the swarm or the number of robots in $\mathcal{R}$ is set to 10, albeit there is a post evolution scalability analysis assessing the task for different swarm sizes.

Initialization: The initialization of the swarm of robots' positions is carried out using a random spatial graph generator. By spatial graph, we refer to the fact that nodes belong to a position in the environment and the graph edges' existance is conditioned to the distance between nodes. The algorithm, which is not described in this Master Thesis because it is out of its scope, allows the formation of random compact spatial graphs with the guarantee that no isolated nodes arise. Thus, the specialization to a particular swarm topology is avoided. Less importantly, the robot orientations are initialized by sampling from $\mathcal{U}(0, 2\pi)$ for each agent.

Sensors, Actuators and Communication: Agents can claim their leadership by means of turning their LED actuator on. As it was explained in Section 3.2.3, the LED actuator is controlled by action a_{LED} generated by the controller's CTRNN. In this scenario, the LED can be either on ($a_{LED} = 1$) or off ($a_{LED} = 0$). Agents can communicate with other robots in

their vicinity using a minimal communication system with local interaction. As we exposed in Section 3.3, the communication sensor can provide diverse communication information, about the message context, depending on the application requirements. Summarizing, the possible communication variables to which robots can access are the received message vector, the signal intensity of the reception, the orientation of the sensor's sector capturing the message (receiving orientation or θ_{RX}), the orientation of the sensor's sector of the robot emitting the message (sending orientation or θ_{TX}) and the communication mode or state. Moreover, as it was stated in the corresponding section, the receiving orientation and the sending orientation are encoded in a 2-dimensional vector (see Eq. 3.15). In this experiment, only the message, the receiving orientation and the communication state are used. The signal strength of the sensed message is not utilized because the graph is static during episodes. The dimension of the message vector is fixed to 2 with $K = 4$ (see Section 3.3), resulting in a total of $2^4 = 16$ different symbols. Therefore, the subsets of sensors and actuators employed in this experiment are the following:

$$\mathcal{S}_{E_A} = \{RX\},\ \mathcal{A}_{E_A} = \{LED, TX\}$$

Neural Controller and evolution hyperparameters: It has been previously said that the unique sensor in this experiment is the RX communication receiver and that the actuators are the LED actuator and the IR transmitter TX. Furthermore, the communication variables are the received message $\mathbf{m}_{RX} \in \mathcal{C}^2$, the communication state ($MODE$) with values in $\{0, 1\}$ and the receiving orientation θ_{RX}, which is encoded as a vector in the unit circumference with values in $[-1, 1]^2$.

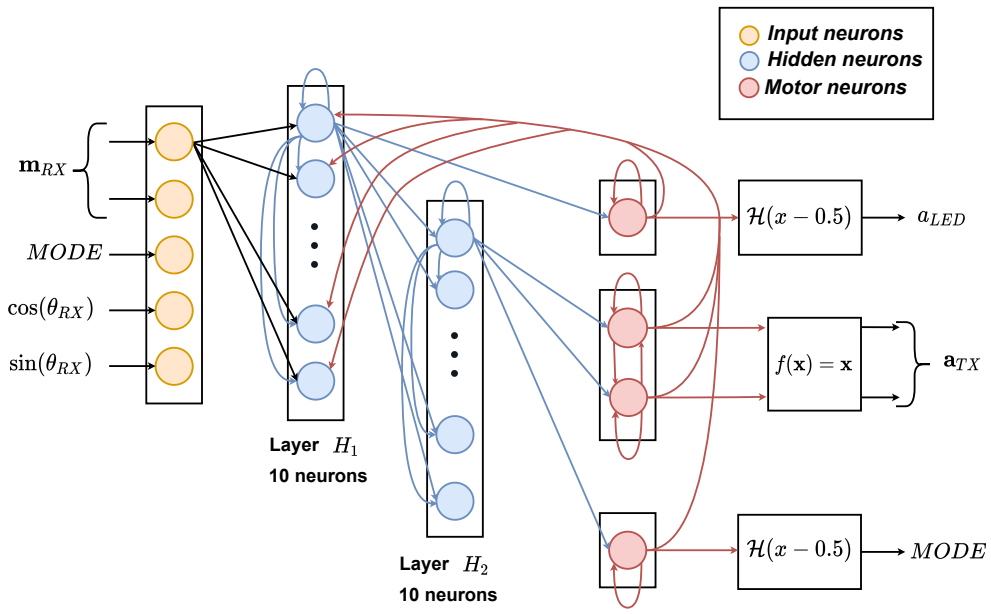


Figure 5.1: CTRNN architecture of the leader identification controller. In order to simplify the diagram, only the connections of the first neuron of each layer are illustrated. The synapses from rest of neurons in the layers have the same post synaptic neurons as the ones depicted for the first unit. The colors of the connection arrows indicate the kind of layer to which the pre synaptic neuron belongs to. Note that input neurons, in yellow, are placeholders of input stimulus and not actual neuron models. The boxes on the right side of the diagram display the firing rate decoding function. $\mathcal{H}$ is the Heaviside function.

Similarly, the output message elaborated by the CTRNN and to be transmitted is $\mathbf{a}_{TX} \in [0,1]^2$, being consistent with the notation exposed in Section 3.3. It should be mentioned that the message $\mathbf{a}_{TX}$ has not been subject to quantization (see Alg. 4) yet. Thereafter, the total stimuli fed to the CTRNN is $\boldsymbol{\phi}(t) \in \mathcal{C}^2 \times \{0,1\} \times [-1,1]^2$ and the total output action vector is $\mathbf{a}(t) \in \{0,1\}^2 \times [0,1]^2$, leading to a CTRNN architecture with 5 input nodes and 4 motor neurons.

Fig. 5.1 shows the CTRNN architecture used in this experiment. The hidden neurons of the CTRNN are partitioned into two ensembles or hidden layers H_1 and H_2 with 10 neurons each. Therefore, the CTRNN has 24 neurons and 5 inputs in total. Note that the hidden layer H_2 adds an additional processing step for the message generation, using the output of H_1. On the contrary, the LED motor neuron is directly connected from H_1. Moreover, all the ensembles have recurrent connections among their neurons and motor outputs are fed back to the neurons in layer H_1, as it can be observed in the figure. An important remark to be pointed out is that connections are randomly created between any pair of neurons i and j with some probability p_c^{ij} (see Section 3.4). The synapses that are not shown in Fig. 5.1 have zero probability by definition of the architecture. In contrast, in a scenario in which all the connection probabilities between any pair of neurons is 1, the architecture would correspond exactly to the one exposed in the diagram of the figure. Table 5.1 gathers the connection probabilities between layer neurons. Note that the feed forward connections are established with probability 1 while the recurrent synapses are created with reduced probabilities. Finally, there are only few details to be defined in relation to the

Presynaptic Layer	Postsynaptic Layer	Connection Probability
Input Layer	Hidden H_1	1
Hidden H_1	Hidden H_2	1
Hidden H_1	Hidden H_1	0.7
Hidden H_2	Hidden H_2	0.7
Hidden H_1	Motor LED	1
Hidden H_2	Motor Message	1
Hidden H_2	Motor Comm. State	1
Motor LED	Hidden H_1	0.85
Motor Message	Hidden H_1	0.85
Motor Comm. State	Hidden H_1	0.85

Table 5.1: Connection Probabilities between layers in the CTRNN architecture of Fig. 5.1

evolutionary optimization hyperparameters. The genotype length resulting from the random architecture generation is 403. This number incorporates the synapse strengths and the time constants, biases and gains of neurons. The number of evaluation time steps fixed for this experiment is of 300 time instants, which is a sufficient amount of time for efficient task completion (see Chapter 6). Each genotype is evaluated in 5 independent trials with different random initialization, as a tradeoff between costly evaluation and fitness estimate variance. The number of generations in this experiment, for both GA and SNES, is settled to 1000 generations.

Related Work: The leader selection is an interesting problem in swarm robotics as the existence of a leader of the group, or from a more general perspective different assigned roles in the swarm, eases the resolution of multiple collective tasks, such as flocking [14, 97],

foraging [42], coordinated movement [98] or goal navigation [99] among others. The selection of a leader of a swarm of agents can be understood as a role allocation problem with two roles, the leader and the followers. The election of a leader has been treated in several works [30, 31, 32, 55]. Firstly, the leader selection procedure in [30] avoids the usage of direct communication among robots as it only employs the positions of other agents in the neighborhood. A robot becomes a leader whenever all the other robots in its vicinity lie on the same quadrant, considering the robot's position as the origin of coordinates. In [31], the authors designed a voting algorithm based on local communication in a static swarm for the leader election, among other cooperative tasks. The results of their experiments successfully show consensus in the selection in most cases. Alternatively, one of the multiple experiments in [32] is the leader election. They employ Wave Oriented Swarm Paradigm (WOSP) techniques in order to trigger the emergence of collective behaviors, such as leader election, with local binary information exchange. The controllers are also handcrafted as outstandingly simple and compact finite state machines. Nevertheless, in these papers, the communication information is received from all possible orientations at the same time, as opposed to our experiments, that assess task completion with minimal communication resources. Moreover, neither evolution nor neural controllers are explored in these papers. On the contrary, in [55], the leader selection and role allocation problems are faced using neural controllers and evolutionary computation. The swarm members communicate locally through a communication system and a robot can assume the role of leader by directly maximizing its communication output.

Fitness Function: The fitness function rewards two main aspects of the swarm behavior in this problem, namely, leader identification and its preservation. On the one hand, the instantaneous fitness score increases at each time step when there is uniquely one leader in the group. On the other hand, the reward rises proportionally to the number of consecutive time steps that the leadership has been assumed by the same robot. The fitness function at each simulation instant is exposed in Eq. 5.2.

$$f_A(\mathscr{G}, k) = \mathbf{w}(k)^\top \mathbf{a}_{LED}(k)\, \delta\left(1 - \sum_{r \in \mathcal{R}} \mathbf{a}_{r,LED}(k)\right) \tag{5.2}$$

The vector $\mathbf{a}_{LED}(k)$ gathers the LED actions performed by all the agents in the swarm at time step k, subject to genotype $\mathscr{G}$. Moreover, $\mathbf{a}_{r,LED}(k)$ is the scalar LED action of the robot r in the group (which is one of the components of $\mathbf{a}_{LED}(k)$). The vector $\mathbf{w}(k)$ is a credit assignment vector that represents the number of consecutive time steps that a robot has been the leader.

$$\mathbf{w}(k) = \begin{cases} \min\{\mathbf{w}(k-1) + 0.1\,\mathbf{a}_{LED}(k),\, 5\} & \text{If } \sum_{r \in \mathcal{R}} \mathbf{a}_{LED}(k, r) = 1 \\ & \quad \text{and } \mathbf{w}(k-1)^\top \mathbf{a}_{LED}(k) > 0 \\ \underbrace{(0, \ldots, 0)}_{R \text{ times}}{}^\top & \text{Otherwise} \end{cases} \tag{5.3}$$

Specifically, Eq. 5.3 describes the computation of $\mathbf{w}(k)$. Note that the components corresponding to activated LEDs are increased 0.1 up to a maximum of 5. This increment is only performed if there is only one leader (i.e. only one LED action to 1) and the current leader is the same as the previous one, as specified in the condition of the equation. Provided that none of the former conditions are fulfilled, the credit assignment vector is reset to zeros. With the meaning of $\mathbf{w}(k)$ in mind, we can resume the explanation of Eq. 5.2. Clearly, the dot product $\mathbf{w}(k)^\top \mathbf{a}_{LED}(k)$ will be higher as the credit assignment of a leader grows. Moreover, the leadership of a single agent is verified using the Kronecker delta $\delta(z)$, whose result is one if and only if $z = 0$, and zero otherwise.

The fitness score of a trial is computed as the temporal average of the instantaneous

fitness value of each evaluation time step:

$$F_A = \frac{1}{T_E} \sum_{k=1}^{T_E} f_A(\mathscr{G}, k)$$

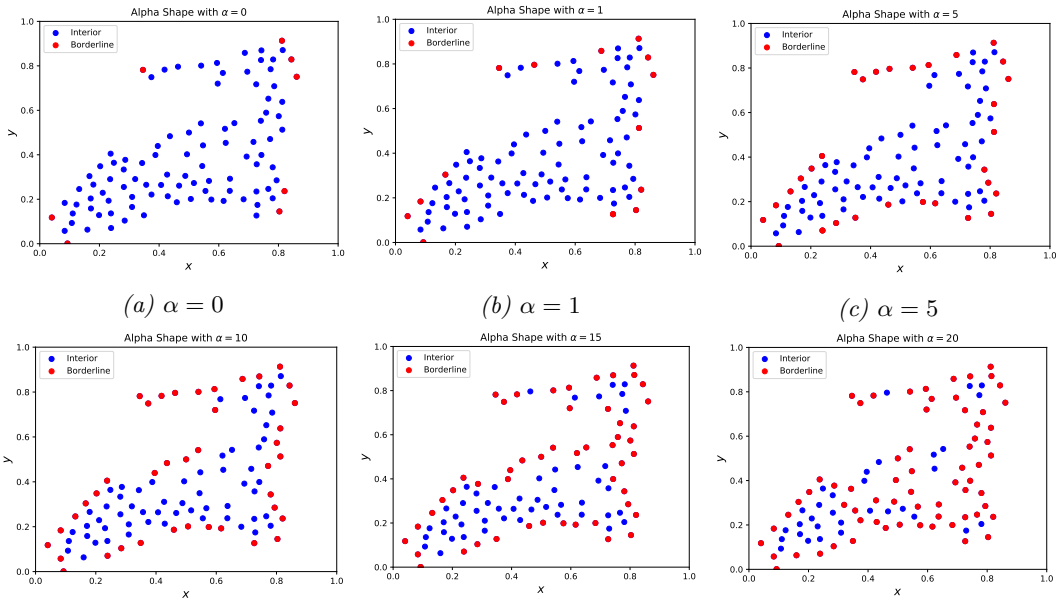
The minimum achievable fitness score is 0. Similarly, recalling that a leader fault occurs after 50 consecutive time steps of leadership, the maximum achievable fitness with a simulation duration T_E of 300 time steps is:

$$F_A^* = \frac{6}{300} \sum_{i=0}^{49} \min\{0.1\, i,\, 5\} = \frac{6 \cdot 122.5}{300} = 2.45$$

However, reaching the maximum score is highly difficult in practice, as it would require a perfect leadership and an instantaneous recovery after leader fault. Finally, each genotype is evaluated 5 independent times so that multiple fitness samples are used to construct the final expected fitness estimate (see Eq. 5.1).

5.2 Experiment B: Borderline Identification

Task Description: Borderline or frontier identification of a swarm of robots is the problem of detecting which nodes in a cluster surround the others. More formally, given a set of points $P \subset \mathcal{T}$, the aim is to find the subset of these points P_B such that there is a closed polygon joining all elements in $P_B \subset P$ whose area contains all the other points in $P \setminus P_B$. This boundary is approximated, for arbitrary and randomly generated node distributions,



(a) $\alpha = 0$ *(b)* $\alpha = 1$ *(c)* $\alpha = 5$

(d) $\alpha = 10$ *(e)* $\alpha = 15$ *(f)* $\alpha = 20$

Figure 5.2: Comparative example of alpha shape algorithm for different values of α *and 100 points.* $\alpha = 0$ *results in the convex hull solution, increasing* α *leads to more realistic borderlines.*

using the alpha shape algorithm, as a generalization of convex hulls (see [100]). In order to understand the meaning and importance of the parameter α, Fig. 5.2 shows the alpha shape

of a randomly generated set of 100 points with different values of the parameter α. When α is zero, the alpha shape is the same as the convex hull (see Fig. 5.2a). Increasing α leads to more realistic and well defined shapes of the border. Large values of the parameter can lead to incorrect shapes as exemplified in Fig. 5.2f. After several tests for different point distributions and multiple number of points, we heuristically concluded that a value of $\alpha = 15$ is suitable for our mobile robot application. The borderline points (P_B) are those points belonging to the obtained alpha shape. The interior points are the remaining nodes in the swarm, that are not considered extremities. Only those agents belonging to the alpha shape of the swarm should identify themselves as borderline. Therefore, at each simulation cycle, the experiment can be understood as a binary classification problem in which agents have to decide if they belong to the boundary or not. Alpha shape is used to generate target values that increase the fitness of the swarm if classification is correct. In the experiment proposed here, the target alpha shape is the same during the entire simulation period because of the stationarity of the swam. In practice, the lack of a large number of points, in our case robots, leads to highly probable scenarios where the borderline polygon is not closed (for instance, a swarm forming a line or a V-shape). In this case, all the robots in the swarm would belong to the frontier ($P_B = P$).

Initialization: The initialization of this experiment is exactly the same as the one described in Experiment A (see Section 5.1). The random graph initialization allows the reaching of a generalized solution capable of detecting the border regardless of the swarm distribution.

Sensors, Actuators and Communication: The sensors, actuators and communication of the borderline identification task are the same as in the leader selection experiment (Experiment A). The main difference to be pointed out is that a robot uses the binary LED action to announce its decision of identifying itself as a frontier member within the group (by turning the LED on). The communication setup is also shared with Experiment A, using the same communication information and maintaining the message dimension as 2. The sets of enabled sensors and actuators are:

$$\mathcal{S}_{E_B} = \{RX\},\ \mathcal{A}_{E_B} = \{LED, TX\}$$

Neural Controller: Similar to the previous building block, the neural controller is exactly the same as in Experiment A with the CTRNN architecture exposed in Fig. 5.1 due to the use of the same sensors and actuator. The number of layers, neurons per layer and connection probabilities (see Table 5.1) are also preserved. Furthermore, the same seed is employed for creating the synapses and, consequently, the unweighted adjacency matrix is a copy of the one sampled in the leader selection task. Regarding the evolution hyperparameters, there are 5 independent evaluation trials, 1000 generations and 300 simulation time steps per trial.

Related Work: The task of borderline identification of a swarm is a relevant problem in the context of collective task such as flocking, aggregation or shape formation. The awareness of the limits of the swarm can refine these task and potentially aid the avoidance of swarm disconnection. As far as we are concerned, only Varughese et al [32] have treated the problem of boundary identification in the context of swarm robotics. They use minimalistic robot behaviors and binary communication using Wave Oriented Swarm Paradigm (WOSP).

Fitness Function: At each time step, the instantaneous fitness function will be higher as the number of correct LED decisions increase. A correct decision means that either a borderline member activates its LED (true positive) or that an interior robot turns off its LED (true negative). Let $\mathcal{R}_B \subset \mathcal{R}$ be the subset of robots belonging to the swarm borderline or frontier (note that there is a bijective correspondence between the elements of $\mathcal{R}_B$ and

P_B). On the contrary, the subset of robots belonging to the interior of the swarm is denoted as $\mathcal{R}_I = \mathcal{R} \setminus \mathcal{R}_B$. The instantaneous fitness function is displayed in Eq. 5.4. The function has two terms that are combined as a product. The first term corresponds to the number of borderline robots successfully identified divided by the number of borderline members. The second part is the number of correct classifications of interior agents divided by the number of interior robots. R_B and R_I denote the cardinality of the sets $\mathcal{R}_B$ and $\mathcal{R}_I$, respectively.

$$f_B(\mathscr{G}, k) = \frac{1}{R_B\, R_I} \left(\sum_{r \in \mathcal{R}_B} \mathbf{a}_{r,LED}(t) \right) \cdot \left(R_I - \sum_{r \in \mathcal{R}_I} \mathbf{a}_{r,LED}(t) \right) \tag{5.4}$$

From a different perspective, the terms measure separately the true positive rate (TPR) and the true negative rate (TNR) instead of computing the directly the total accuracy of the swarm. This fitness breakup is principally performed because the number of borderline and interior agents (R_B and R_I) are highly unbalanced in most of the random initializations of the swarm topology. This leads to naive and utterly suboptimal solutions in which all robots identify themselves as interior or borderline due to the high overall accuracy. As in all the experiments, the fitness score of a trial is computed as the average of the instantaneous fitness values of each time step:

$$F_B = \frac{1}{T_E} \sum_{k=1}^{T_E} f_B(\mathscr{G}, k)$$

Similarly, 5 trials are accomplished, computing the final expected fitness estimate as the sample mean of all the episode fitness scores. The fitness maximum and minimum achievable values are 0 and 1 respectively.

5.3 Experiment C: Orientation consensus

Task Description: The objective of this experiment is to evolve a cooperative behavior of a group of robots capable of solving the problem of orientation consensus. By orientation consensus we refer to the task in which all the robots in a group have to point to the same direction. Thereafter, the orientations of all robots $\theta_r(t)$ have to converge to the same value for reaching the maximum fitness. The positions of the agents are fixed during the evaluation period so that the overall swarm graph is always the same. Alternatively, robots can alter their orientation by means of rotation movements at an angular speed modulated by the neural controller. Robots do not have any common absolute reference (for instance be a light source or a compass) that would utterly ease the orientation consensus achievement. Agents must infer the orientation of their neighbors relative to its own orientation merely using the minimal communication system exposed in Section 3.3, which makes it a challenging experiment.

Initialization: The initialization of this experiment is exactly the same as the one described in Experiment A (see Section 5.1). Furthermore, random uniform initialization of the agents makes it impossible for the robot's CTRNN to memorize the orientation of other individuals based on the initialization. Therefore, as in the other experiments, the initialization promotes the emergence of solutions that highly generalize for any starting conditions of the system.

Sensors, Actuators and Communication: The only sensor to be used in this experiment is, as in the previous ones, the communication receiver. However, in addition to the message vector and the receiving orientation θ_{RX}, the orientation or sector from which the message was transmitted by the neighboring robot sender, θ_{TX}, is also provided (see Section 3.3 for more details). We found that this variable is essential in order to reach acceptable results in this experiment, under the defined experimental setup and minimal communication means. The main reason is that agents must discover the heading orientation

of, at least, its neighborhood relative to its own orientation. Alternatively, the enabled actuators are the communication transmitter (TX) and the wheel actuator (WA). In order to restrict movements to rotations, the corresponding CTRNN output only controls the right wheel through scalar action $a_{wr} \in [-1, 1]$ (which is scaled by the maximum wheel rotation ω_{max} afterwards). Thereafter, the final wheel action vector is constructed as $\mathbf{a}_{WA} = (a_{wr}, a_{wl})^\top = (a_{wr}, -a_{wr})^\top$ so that only rotation movements can be accomplished. The sign of the action defines whether the spin is clockwise or counterclockwise. Additionally, the absolute value of the action defines the rotation velocity in the sense specified by the sign. Finally, regarding the communication parameters, the message dimension is fixed to 2 and $K = 4$, leading to 16 possible symbols. The sets of enabled sensors and actuators are:

$$\mathcal{S}_{E_C} = \{RX\},\ \mathcal{A}_{E_C} = \{\text{WA},\ TX\}$$

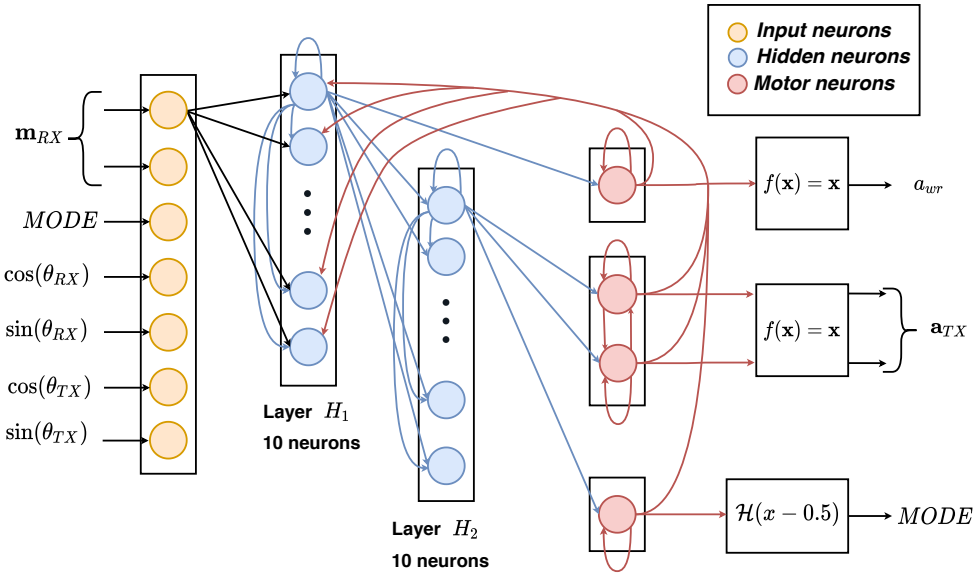


Figure 5.3: CTRNN architecture of the orientation consensus controller. In order to simplify the diagram, only the connections of the first neuron of each layer are illustrated. The synapses from rest of neurons in the layers have the same post synaptic neurons as the ones depicted for the first unit. The colors of the connection arrows indicate the kind of layer to which the pre synaptic neuron belongs to. Note that input neurons, in yellow, are placeholders of input stimulus and not actual neuron models. The boxes on the right side of the diagram display the firing rate decoding function. $\mathcal{H}$ is the Heaviside function.

Neural Controller and evolution hyperparameters: Figure 5.3 shows the CTRNN architecture used in this experiment. As it can be observed, it is highly similar to the architecture displayed in Fig. 5.1, excluding the input and motor layers. The stimuli vector fed to the CTRNN is of dimension 7. It is formed by the communication information described in the previous paragraph, with the addition of the encoded θ_{TX}, taking values in $[-1, 1]^2$. The overall stimuli vector is $\boldsymbol{\phi}(t) \in \mathcal{C}^2 \times \{0, 1\} \times [-1, 1]^4$. There are a total of 4 motor neurons, devoted to the generation of the right wheel action ($a_{wr} \in [-1, 1]$), the message vector

($\mathbf{a}_{TX} \in [0,1]^2$) to be transmitted and the next communication state (in $\{0,1\}$). Thus, the total action vector is $\mathbf{a}(t) \in [-1,\ 1] \times [0,\ 1]^2 \times \{0,\ 1\}$. The decoding of the motor neuron's firing rate into actions is the identity mapping in the case of the wheel action and the message content. The heaviside function centered at 0.5 is alternatively used to decode the communication state $MODE$. Furthermore, in order to produce neuron outputs, of the wheel motor neuron, in the interval $[-1,\ 1]$, the activation of the neuron controlling action a_{wr} is the tanh function. The connection probability between layers is as in Table 5.1 with a slight alteration to fit the current architecture. Specifically, Motor LED is replaced by Motor Right Wheel in rows 5 and 8, maintaining the same connection probabilities.

The total genotype length after the random synapse creation is fixed to 435. The genotype evaluations are executed during 600 time steps and a total of 5 trials per evaluations are established to compute the sample fitness estimate. The number of generations is settled to 1000 generations.

Related Work: Orientation consensus is an important and widely explored cooperative task as it is one of the pillars of flocking behaviors according to Reynolds' rules [11]. Therefore, the problem of heading alignment has been principally studied and assessed in the context of flocking experiments. Heading alignment is addressed in [13] for self-organized flocking in mobile robot swarms using a virtual heading sensor. Each robot senses its own orientation with respect to the North reference, using a digital compass, and broadcasts it to its neighborhood. In [15], the authors propose heading alignment behavior in which a only subset of robots, called informed, are aware of a common objective direction. Informed agents communicate goal direction to its neighboring robots while uninformed agents relay the average incoming message from its vicinity. Robots correctly achieve alignment with their heading pointing to the goal direction. The swarm members know an absolute reference throughout measuring the light intensity emitted by a light source. In a more recent work, the authors of [12] successfully evolve neural controllers for flocking behaviors. Their fitness is composed by cohesion, separation and alignment terms. Focusing on alignment, robots have an alignment sensor that measures its orientation relative to the average orientation of its neighborhood. Additionally, in this work we focus on the limits of minimal local direct communication in which robots do not share any absolute reference. This conditions are similar to those used in [43], where an evolutionary algorithm optimizes the parameters of a recurrent neural network. The emerged behavior employed a situated communication as it did not harness the message information itself but the physical conditions of the communication.

Fitness Function: The fitness function is composed by two terms that are merged in a multiplicative manner. The instantaneous fitness function is shown in Eq. 5.5.

$$f_C(\mathscr{G}, k) = \frac{1}{R} \sum_{r \in \mathcal{R}} \left(1 - \frac{\min\{2\pi - |\theta_r(k) - \overline{\theta}(k)|, |\theta_r(k) - \overline{\theta}(k)\}}{\pi} \right) \cdot (1 - \|\mathbf{a}_{wr}(k)\|_1) \tag{5.5}$$

The first term measures the mean orientation deviation or misalignment of each robot with respect to the mean orientation. The mean orientation is defined as:

$$\overline{\theta}(t) = \frac{1}{R} \sum_{r \in \mathcal{R}} \min\{\theta_r(t), 2\pi - \theta_r(t)\} \tag{5.6}$$

Note that, the term has a lower limit of 0 because of the division by π, which is the maximum orientation misalignment.

The second part of the fitness function rewards robots for reducing their rotation velocity. In order to punctuate this behavioral property, the vector of right wheel actions at each time step $\mathbf{a}_{wr}(k)$ is considered. $\mathbf{a}_{wr}(k)$ is a vector of dimension equal to the swarm size that collects the actions controlling right wheel of all the robots. Consequently, the L_1 norm is directly used to map robot velocities to fitness scores. The motivation behind this term is

that, after several evolution optimization attempts, the swarm was capable of solving the task by matching the vicinity's phase and rotation speed. Thereafter, the entire swarm was approximately aligned, but the rotation was utterly appreciable.

5.4 Experiment D: Light follower

Task description: As the last experiment to be explored in this Master Thesis, the light following task is presented. As the name indicates, it consists in following a mobile light source as closely as possible. The experiment is called light follower because the target entity is a mobile light source. However, the task can be generalized to the object following problem. Moreover, the entity to be followed can be another robot acting as, say, leader of the swarm. In order to hinder the experiment, the following constrain is applied: only a small subset of robots in the swarm are capable of sensing the light source within a range of vision. The rest of the agents are unable to perceive the presence and location of the light. Hereafter, we denote the former subset of robots as photosensitive agents and the latter group as non-photosensitive robots. The sole mechanism of non-photosensitive agents to accomplish the task is throughout communication with photosensitive robots, either abstract or situated communication. During the optimization stage, only two agents are granted with the ability of measuring input light source. Although the remaining robots are equipped with light sensors, their measurement is always zero.

The movement of the light source is a simple orbit around the central coordinates $(W/2, H/2)$ of the torus. The kinematics of the light are, therefore, summarized in Eqs. 5.7 and 5.8. We express the position update in polar coordinates ϑ and r dependent on the simulation cycle k:

$$\left.\begin{aligned} \vartheta(k+1) &= \vartheta(k) + \gamma(k)\,\Delta\vartheta \\ \mathsf{r}(k+1) &= \mathsf{r}(k) + \frac{(R - \mathsf{r}(k))}{50} \end{aligned}\right\} \tag{5.7}$$

where $\gamma(k) \sim \mathcal{B}(10^{-2})$ is a Bernoulli random variable that states the rotation sense of the light at each time step. Thereafter, with a very small probability, the orbit sense is inverted. $\Delta\vartheta$ represents the angular displacement of the light's position within the orbit at each time step k. This constant increment was fixed to 0.01 radians per simulation step. The radius r is initialized at zero and tends to the constant value $R = 2$m as time steps are elapsed. The previous polar coordinates are converted into a cartesian position of the light as in Eq. 5.8, where the center of the orbit is clearly established.

$$\mathbf{x}_\ell(k+1) = \mathsf{r}(k+1) \begin{pmatrix} \cos(\vartheta(k+1)) \\ \sin(\vartheta(k+1)) \end{pmatrix} + \begin{pmatrix} W/2 \\ H/2 \end{pmatrix} \tag{5.8}$$

The initial values of the light dynamics are always fixed as,

$$\vartheta(0) = 0, \mathsf{r}(0) = 0 \Rightarrow \mathbf{x}_\ell(0) = \left(\frac{W}{2}, \frac{H}{2}\right)^\top$$

In short, the light source is initialized at position $(W/2,\ H/2)$ and it follows a spiral trajectory, with increasing radius, until $\mathsf{r}(k) \approx R$. Afterwards, it orbits around the previously mentioned initial position, with a low probability of inverting is rotation sense.

Initialization: The initialization of the robot's positions is addressed by randomly sampling from a square centered at the center of the torus. More precisely, the sampling of the position of each robot r is as follows,

$$\mathbf{x}_r(0) = (x(0), y(0))^\top \Rightarrow x(0) \sim \mathcal{U}\left(\frac{W-A}{2}, \frac{W+A}{2}\right),\ y(0) \sim \mathcal{U}\left(\frac{H-A}{2}, \frac{H+A}{2}\right)$$

where $\mathcal{U}$ is the uniform distribution and W, H are the dimensions of the flattened torus (see Section 3.1), which are fixed to 10m. A is the length of the sides of the square from where positions are initialized. Alternatively, orientations are sampled uniformly from $\mathcal{U}(0, 2\pi)$.

Sensors, Actuators and Communication: The sensors enabled for this experiment are the ambient light sensors and the communication receiver. The communication information harnessed is the 2 dimensional message, the communication state, the orientation from where the message was sensed and, finally, the signal intensity. Recall that signal intensity is called ϕ_{DS} because it actually uses the IR distance sensor measurement in the direction of the incoming message. Moreover, we do not include the entire distance sensor measurement, so that the CTRNN can only be aware of one neighbor at the same time. This restriction is imposed because we want to constrain as much as possible the environment information and swarm interactions. In addition to the communication information, the light sensor produces a vector of 6 components, corresponding to the instantaneous measurement in each of the 6 sectors. The actuators that allow agent interaction with the environment and with other robots are the wheel actuator and the communication transmitter. In contrast to orientation consensus experiment, the wheel actuator controls both right and left wheels as both rotation and translation movements are allowed. The sets of enabled sensors and actuators are:

$$\mathcal{S}_{E_D} = \{RX, LS\}, \mathcal{A}_{E_D} = \{\text{WA}, TX\}$$

Neural Controller and evolution hyperparameters Fig. 5.4 depicts the precise neural architecture used in this experiment.

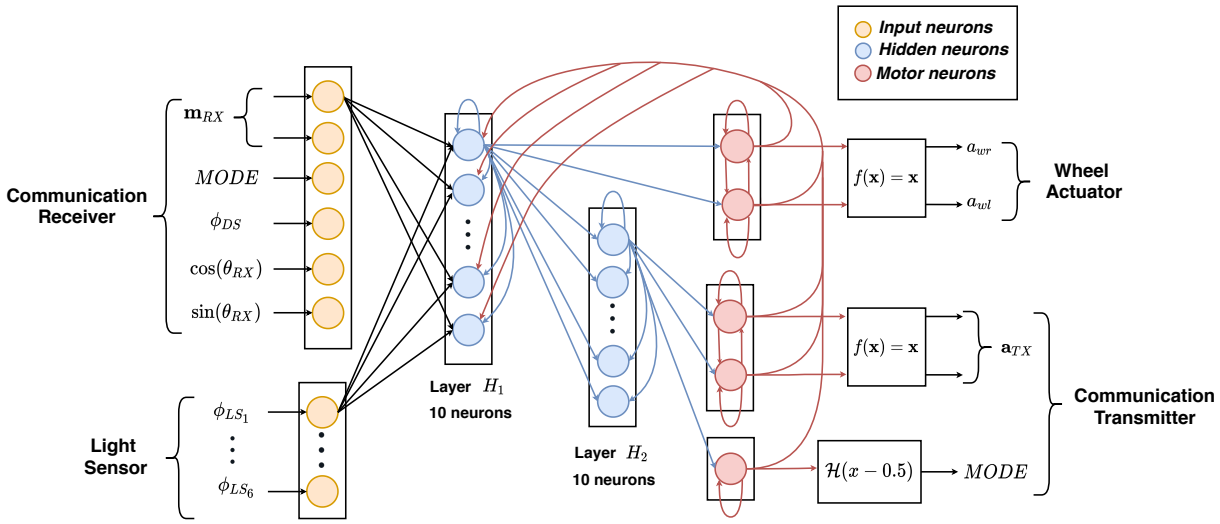


Figure 5.4: CTRNN architecture of the light follower controller. In order to simplify the diagram, only the connections of the first neuron of each layer are illustrated. The synapses from rest of neurons in the layers have the same post synaptic neurons as the ones depicted for the first unit. The colors of the connection arrows indicate the kind of layer to which the pre synaptic neuron belongs to. Note that input neurons, in yellow, are placeholders of input stimulus and not actual neuron models. The boxes on the right side of the diagram display the firing rate decoding function. $\mathcal{H}$ is the Heaviside function.

The principal distinction with former architectures is that a new input layer is added, representing light intensity stimuli to be fed. This layer has 6 nodes, each corresponding to the light intensity measured in one sector of the light sensor. The overall light intensity stimuli $\boldsymbol{\phi}_{LS}$ is bounded in $[0,1]^6$. The input nodes wherein light measurements are injected are connected to the neurons of hidden ensemble H_1. Regarding the input layer devoted to communication receiver information, the signal strength being measured $\phi_{DS} \in [0,1]$ is included. We simplify the notation of this variable, as introduced in Sec. 3.2.2, to denote the signal strength measured by the distance sensor in sector corresponding to angle θ_{TX}. The total number of input nodes is 12 as indicated in the figure, leading to the final stimuli vector $\boldsymbol{\phi}(t) \in \mathcal{C}^2 \times \{0,\,1\} \times [-1,\,1]^2 \times [0,1]^7$. The hidden layers H_1 and H_2 are remained untouched and only connections with other ensembles differ from previous sections. Lastly, the motor neurons generate actions corresponding to the elaborated message $\mathbf{a}_{TX} \in [0,\,1]^2$, the communication mode $MODE \in \{0,1\}$ (with 0 corresponding to relay and 1 to send modes) and the left and right wheel actions $\mathbf{a}_{\mathrm{WA}} = (a_{wr}, a_{wl})^\top \in [-1,\,1]^2$. Then, the total action vector is $\mathbf{a}(t) \in [-1,\,1]^2 \times [0,\,1]^2 \times \{0,\,1\}$ (5 motor neurons). Table 5.2 gathers the connection probabilities of neurons between layers.

Presynaptic Layer	Postsynaptic Layer	Connection Probability
Input Communication	Hidden H_1	1
Input Light	Hidden H_1	1
Hidden H_1	Hidden H_2	1
Hidden H_1	Hidden H_1	0.7
Hidden H_2	Hidden H_2	0.7
Hidden H_1	Motor Wheels	1
Hidden H_2	Motor Message	1
Hidden H_2	Motor Comm. State	1
Motor Wheels	Hidden H_1	0.85
Motor Message	Hidden H_1	0.85
Motor Comm. State	Hidden H_1	0.85

Table 5.2: Connection Probabilities between layers in the CTRNN architecture of Fig. 5.4

The resulting genotypes have a length of 542. Additionally, the evaluation duration is, opposed to other experiments, of a duration of 1000 time steps. Each genotype evaluation consists of 5 independent episodes and there are 650 generations.

Related work: As it was already mentioned in the task description, the problem of light following can be labelled as object following, more broadly covered in the literature. Baldassarre et al. [40] evolved mobile robots controlled by neural controllers on the task moving towards light sources while preserving group compactness through aggregation. Robots were equipped with ambient light sensors, IR sensors to perceive neighboring robots at the close range and directional microphones and speakers for the wide range interaction. They observed the emergence of a broad variety of formations and coordinated movement strategies to preserve clustering, while moving towards the target light. Several works have treated the problem from the perspective of leader following [14, 30, 97] under the framework of flocking (note that there is an obvious overlap of the references with Section 5.1). A recent example of leader following is presented in [30]. There, after it is elected, a leader navigates towards a goal position. The rest of the robots must follow the leader in order to reach the

goal. The authors of [99] address a similar problem as the one considered here. They optimize ANNs through neuroevolution for the task of swarm navigation to a goal location. Within the swarm, only a subset of leader agents that know the goal location is controlled by an ANN. The navigation of the rest of the group is driven by the Reynolds's rules of flocking. Therefore the task of the leaders is both to move towards the goal location while guiding the other agents. The neural controller of the leader is fed with the relative polar coordinates of the goal and the center of mass of the group. Note that regardless of the similarities in the task formulation, the experiment presented in this Master Thesis involves different objectives and imposes tighter constrains in order to assess the limits of minimal communication.

Fitness function: The fitness function applied at each instant has a unique term as exposed in Eq. 5.9.

$$f_D(\mathscr{G}, k) = \frac{1}{R} \sum_{r \in \mathcal{R}} \mathcal{H}\left(\rho_l - \mathrm{d}_{\mathcal{T}}(\mathbf{x}_r(t) - \mathbf{x}_l(t))\right) \tag{5.9}$$

Essentially, for each robot, the function computes, through the Heaviside function, if the robot is inside of a ball centered at the light position (see Eq. 5.10) and of radius ρ_l.

$$\mathcal{H}\left(\rho_l - \mathrm{d}_{\mathcal{T}}(\mathbf{x}_r(t) - \mathbf{x}_l(t))\right) = \begin{cases} 1, & \text{if } \mathrm{d}_{\mathcal{T}}(\mathbf{x}_r(t) - \mathbf{x}_l(t)) \leq \rho_l \\ 0, & \text{otherwise} \end{cases} \tag{5.10}$$

The value of ρ_l is fixed to 0.8m. Thereafter, the fitness at each evaluation cycle is the mean value of the results from the Heaviside mapping. From a different perspective, the fitness score is proportional to the number of robots inside the ball $\mathrm{d}_{\mathcal{T}}(\mathbf{x}_r(t) - \mathbf{x}_l(t)) \leq \rho_l$. This fitness avoids the emergence of naive individual behaviors in which only the photosensitive agents outperform in the problem. The overall trial fitness value is the temporal average of the instantaneous fitness outcomes:

$$F_D = \frac{1}{T_E} \sum_{k=1}^{T_E} f_D(\mathscr{G}, k)$$

Moreover, in a similar way as in the rest of the tasks, the fitness is evaluated in 5 independent trials and the final expected fitness estimate is obtained as the trial average. Clearly, the fitness is bounded between 0 and 1 in this task, albeit maximum values are nearly impossible.

Chapter 6

Results

In this chapter, the behaviors evolved using the GA and NES algorithms, employing the experimental setup established in the previous chapter, are exhaustively analyzed for each task. For each experiment, a comparison between optimization algorithms will be provided based on their fitness results. Thereafter, we focus on the solution, either from GA or NES, whose communication mechanics are richer and more complex for solving the task. The emerged agent interaction is analyzed and assessed considering the following main aspects:

- **Behavior**: Firstly, the emerged behavior and the resulting performance are discussed. Several figures, showing the swarm actions and the goodness of the solution, are shown to demonstrate the performance. A general overview of the behavior mechanisms is provided, excluding the emerged communication as it is described in a different block.
- **Scalability**: An important feature of multi agent systems in robotics is their ability to scale properly in performance as the swarm size increases. Therefore, in each experiment, the behavior is assessed for different swarm sizes.
- **Robustness**: Another desired property of swarm intelligence systems is the robustness of the system under perturbations or unexpected events during the execution process. The robustness is evaluated by means of introducing an alteration in the task at some point in time during the simulation. Therefore, the agents have to react in response to the environmental change.
- **Communication**: Finally, the emerged communication is described for each problem. Principally, the relevant communication information is figured out, highlighting the type of communication that has emerged (e.g. situated or abstract communication), if any.

In order to present the results in the previous items, tools from descriptive statistics are considered. More precisely, 50 independent trials with random initialization are executed and gathered into a dataset. Afterwards, the dataset with 50 samples is used to support most of the figures and results of the experiments.

6.1 Experiment A: Leader Selection

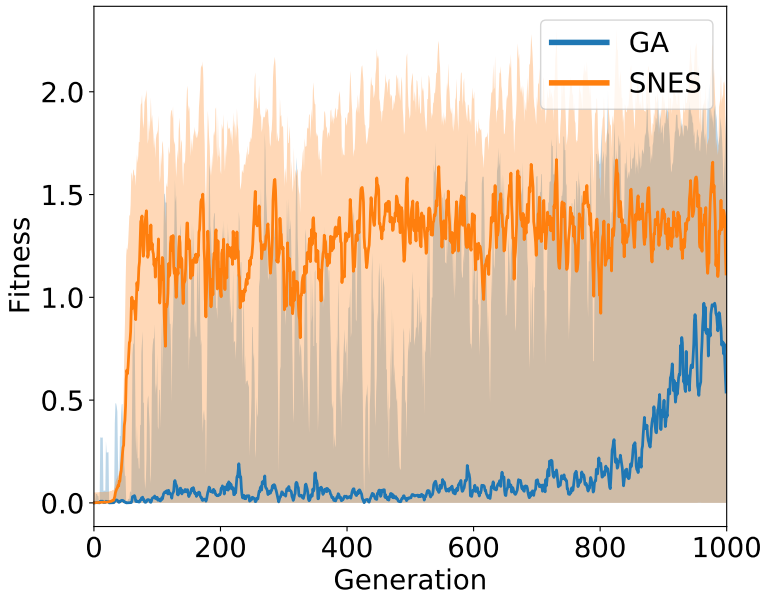


Figure 6.1: Evolution of the fitness function with the generations of GA and SNES in the leader selection task. In each generation, the darker curves are the sample mean fitness scores and the upper and lower contours of the shadow areas represent the maximum and minimum fitness values.

Firstly, Fig. 6.1 illustrates the fitness evolution as generations are elapsed. For each generation, it shows the sample mean fitness score (darker curves) and the maximum and minimum fitness values (upper and lower contours of the shadow areas). The SNES outperforms GA both in terms of the convergence speed and the reached fitness for a large number of generations. Moreover, GA is not able to solve the task correctly at the end of the evolution process. Thus, hereafter, the analysis of the behavior and communication in this experiment is focused on the solution provided by SNES.

Behavior: Genotypes sampled from optimized NES search distributions exhibit an interesting behavior for selecting and preserving leaders of the swarm. We observed two stages during the evaluation process. Firstly, agents carry out a negotiation phase in order to select a leader. Essentially, all the robots claim the leadership at the beginning of the negotiation process by turing their LEDs on. Thereafter, each robot being the leader eventually turns off its LED in order to yield leadership to other members of the swarm. The negotiation process can be understood as a winner takes all competition between agents, so that each potential leader inhibits the others through the communication system. Consequently, the winner of the negotiation, and thus the leader, is the robot whose LED remains turned on. The second phase of the behavior is achieved if an agent is the unique leader of the swarm or, at least, if it does not receive any communication information specifying the opposite. In such a situation, a stable solution is reached and remains still unless a perturbation is introduced in the system. Agents are aware of other robots claiming leadership by means of the communication system. The discussion on the emerged communication is retrieved and elaborated in detail later.

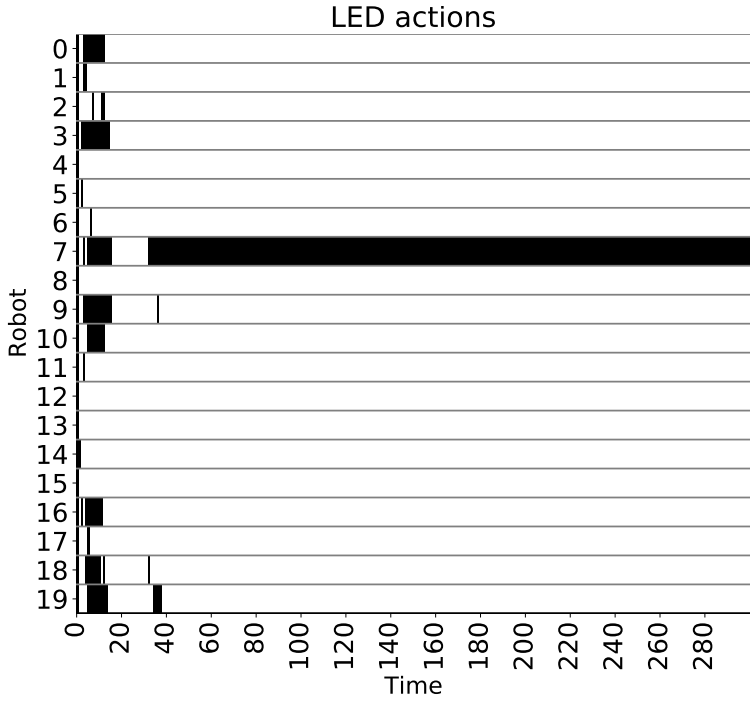
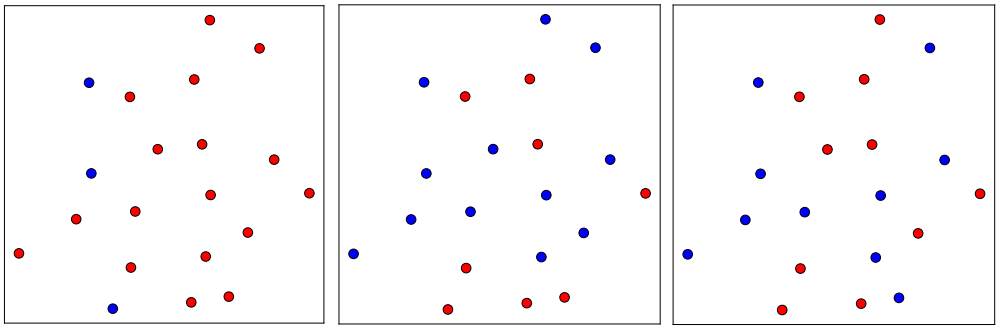


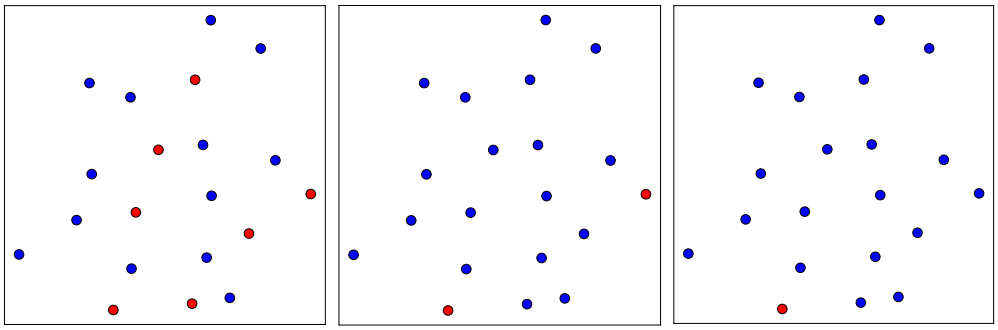
Figure 6.2: LED actions of the robots in a swarm of size 20 as time elapses in the leader selection experiment. Horizontal black bars denote that the LED is activated. Two phases, namely, negotiation and leader settlement, are observed.

Fig. 6.2 shows an example of the emerged behavior, clearly displaying both negotiation and stable leader identification. It shows the LED actions of the agents in a swarm of size 20 as execution time elapses. Black bars indicate that the LED of the corresponding robot in the ordinate axis is turned on. There is no leader fault in this figure as it will be discussed in the robustness analysis below. The results of the figure closely resemble the behavior previously described. The negotiation stage, starting at time step 0 and concluding approximately at simulation cycle 40, exposes the winner takes all process in which LED actions are constantly being deactivated. Nonetheless, the plot shows an unsuccessful negotiation scenario whereby all the robot LEDs are turned off and no leader is selected. In such a case, it can be observed that another negotiation process, of much shorter duration, is initiated with less contestants. Finally, as a result of the second competition, a unique leader is identified and a stable behavior is settled until the simulation ends. The results can be observed in Fig. 6.3 from a different perspective. The figure illustrates different frames of a simulation of this experiment. The red balls represent robots whose LED is turned on and the blue balls show agents with deactivated LED. Only instants previous to time step 20 are shown. After this time instant, leader is settled and stable. The last resource to observe the behavior of robots within this task is provided as a video recoding the simulation [1].

[1] https://youtu.be/ahBlVf9jAbw



(a) Time step 1 *(b) Time step 3* *(c) Time step 5*



(d) Time step 10 *(e) Time step 15* *(f) Time step 20*

Figure 6.3: Snapshots of different time steps of a leader selection simulation. Blue dots represent robots whose LED is deactivated and red balls indicate that the agent's LED is turned on. Figures from (a) to (e) correspond to negotiation phase while in (f) the leader is selected and stabilized.

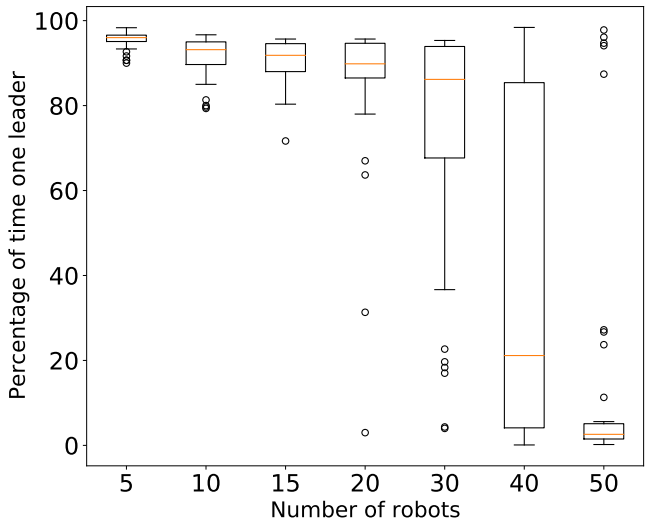


Figure 6.4: Assessment of the scalability capabilities of the evolved solution for the leader election task. For different swarm sizes, the figure shows the distribution of the percentage of the evaluation time that only one robot claims leadership. The sample of size 50 is represented by means of boxplots, where the orange line within the box is the median and each box encloses samples in between the first and third quantiles, also known as the interquartile range. The whiskers extend to the farthest data points that are within 1.5 times the interquartile range. Outliers are shown as white dots.

Scalability: The scalability of the solution is assessed by simulating the experiment with different swarm sizes. Moreover, instead of depicting the results for a single trial, as in the performance evaluation, we collect a total of 50 independent simulations in order to gather a statistically significant number of observations. Note that multiple trials are required for the generation of reliable results because there is randomness in the initialization process. Fig. 6.4 shows the impact of the swarm size on the leader selection experiment. For each of the 50 episodes, the percentage of the total simulation time in which just a single robot claims leadership is computed and represented in boxplots. There is a clear degradation as the number of robots increase. Moreover, this degradation is not only in terms of median degradation but also in terms of interquartile range increase. In general, the SR system for the leader selection experiment scales correctly as the number of robots grow, up to swarm sizes of about 30 agents.

Robustness: In order to verify the system ability to react to unexpected perturbations during runtime, we incorporate the leader fault already introduced in Section 5.1. A leader becomes a faulty robot if it has been the unique leader during 50 consecutive time steps. A defective agent cannot claim leadership nor send its own messages. It can only relay incoming messages from its neighborhood in order to preserve graph compactness.

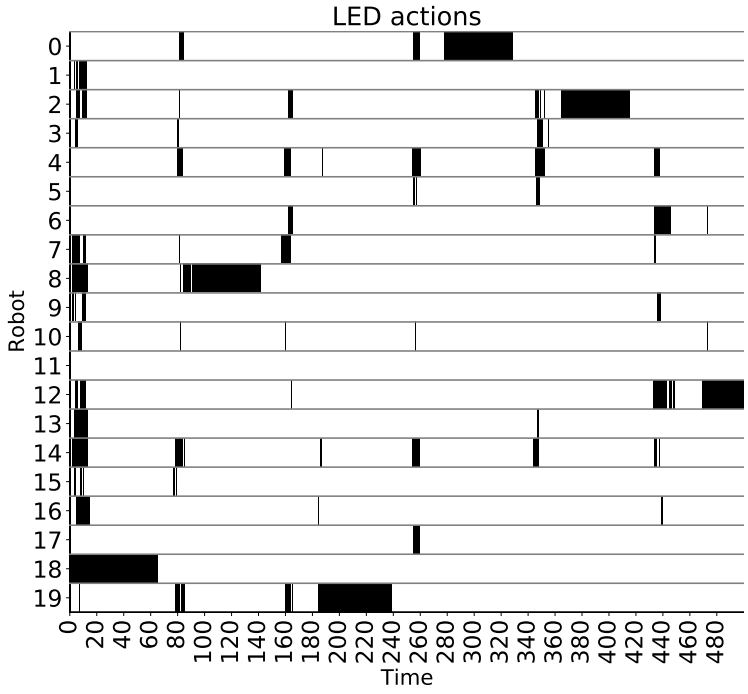


Figure 6.5: LED actions of the robots in a swarm of size 20 as time elapses in the leader selection experiment with leader failure. Horizontal black bars denote that the LED is activated. Two phases, namely, negotiation and leader settlement, are observed.

Firstly, Fig. 6.5 shows the LED actions of each robot with failure perturbations. Again, black bars denote activated LEDs while blank spaces mean that the LED is turned off. It can be observed that agents are totally capable of reacting against previous leader fault. In this situation, an additional phase can be added to the negotiation and stable leader stages. More

precisely, after robot failure, there is a time gap of approximate duration of 10 time instants when all the robots are silent before noticing leader disconnection. Summarizing, in the shown simulation example, robots start with a negotiation as usual. After leader is settled and 50 time steps have elapsed without other leadership claims, the leader fails and the silent period starts. Thereafter, another negotiation is initiated and the process is repeated.

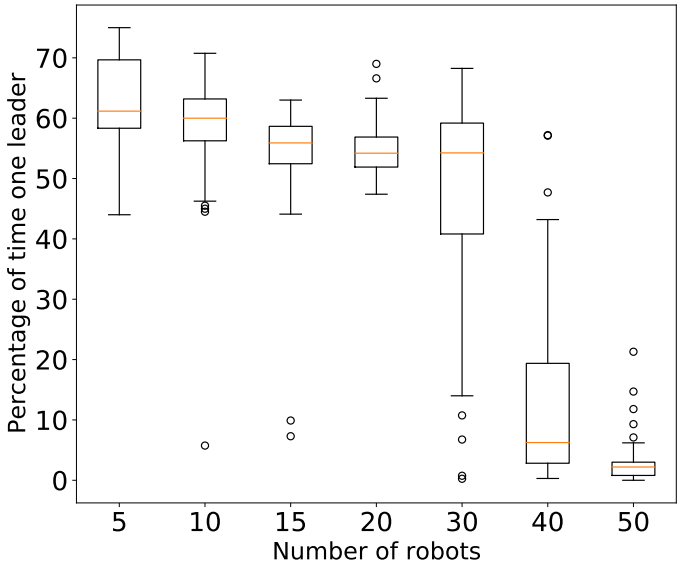


Figure 6.6: Assessment of the scalability capabilities of the evolved solution for the leader election task with leader fault. For different swarm sizes, the figure shows a sample of the distribution of the percentage of the total evaluation time that only one robot claims leadership. The sample of size 50 is represented by means of boxplots, where the orange line within the box is the median and each box encloses samples in between the first and third quantile, also known as the interquartile range. The whiskers extend to the farthest data points that are within 1.5 times the interquartile range. Outliers are shown as white dots.

Fig. 6.6 displays the boxplots of the percentage of time with a single leader in the case of leader fault. All the median estimates are considerably decreased with respect to the ones in Fig 6.4. Furthermore, the results start to decrease drastically with more than 30 robots. It should be mentioned that the general worsening of the performance is acceptable owing to the fact that there is a negotiation and a silent period per leader switch. In fact, assuming that the silent period is 10 time steps and negotiation phase lasts 20 time steps (if successfully accomplished), then every 50 time instants of leadership there are 30 unavoidable time steps of leader reelection. Consequently, the results shown in the boxplots are much more outstanding considering this fact. In the light of the previous figures, it can be stated that the solution reached by SNES in the leader selection task correctly fulfills the robustness requirement (under the imposed test).

Finally, Fig. 6.7 displays the temporal evolution of the number of agents claiming leadership with and without leader fault. Note that in the negotiation phases, corresponding to the peaks, there is a drastic rise of the potential leaders followed by an exponential decay due to the winner takes all competition. The leadership stage corresponds to the curve regions with a single stable leader.

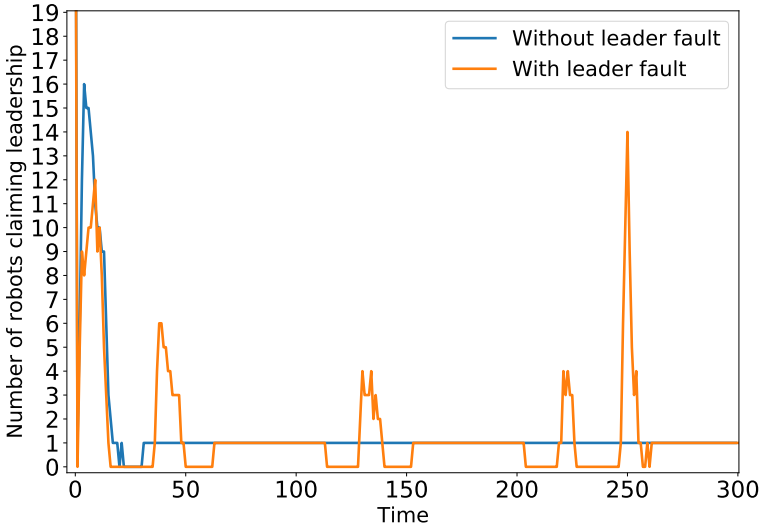


Figure 6.7: Temporal evolution of the number of robots claiming leadership for a trajectory in normal conditions and with leader failure.

Communication: To complement the behavior mechanics previously exposed and analyzed, the communication procedures that emerged from evolution to solve the task are studied. Firstly, in order to gain a general insight on the input variables that are actually harnessed to accomplish leader identification, we performed the following test: for each communication input being fed to the CTRNN, we inhibited the input variable and observed the consequent results. The process to inhibit neural stimulus is straightforwardly accomplished by replacing the input by zeros.

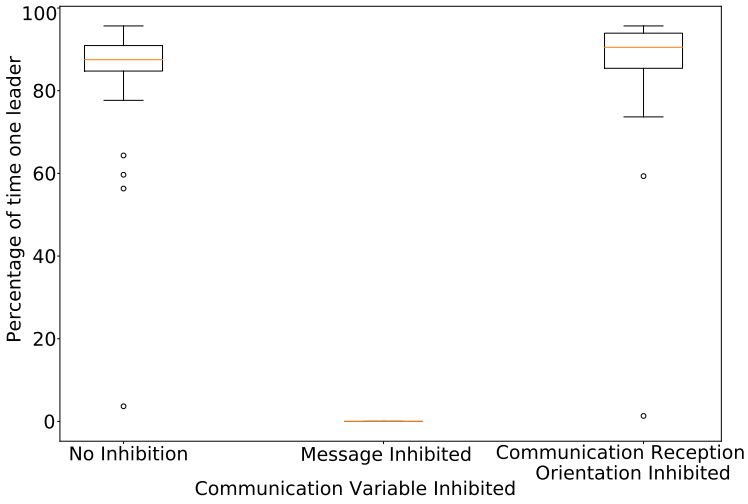
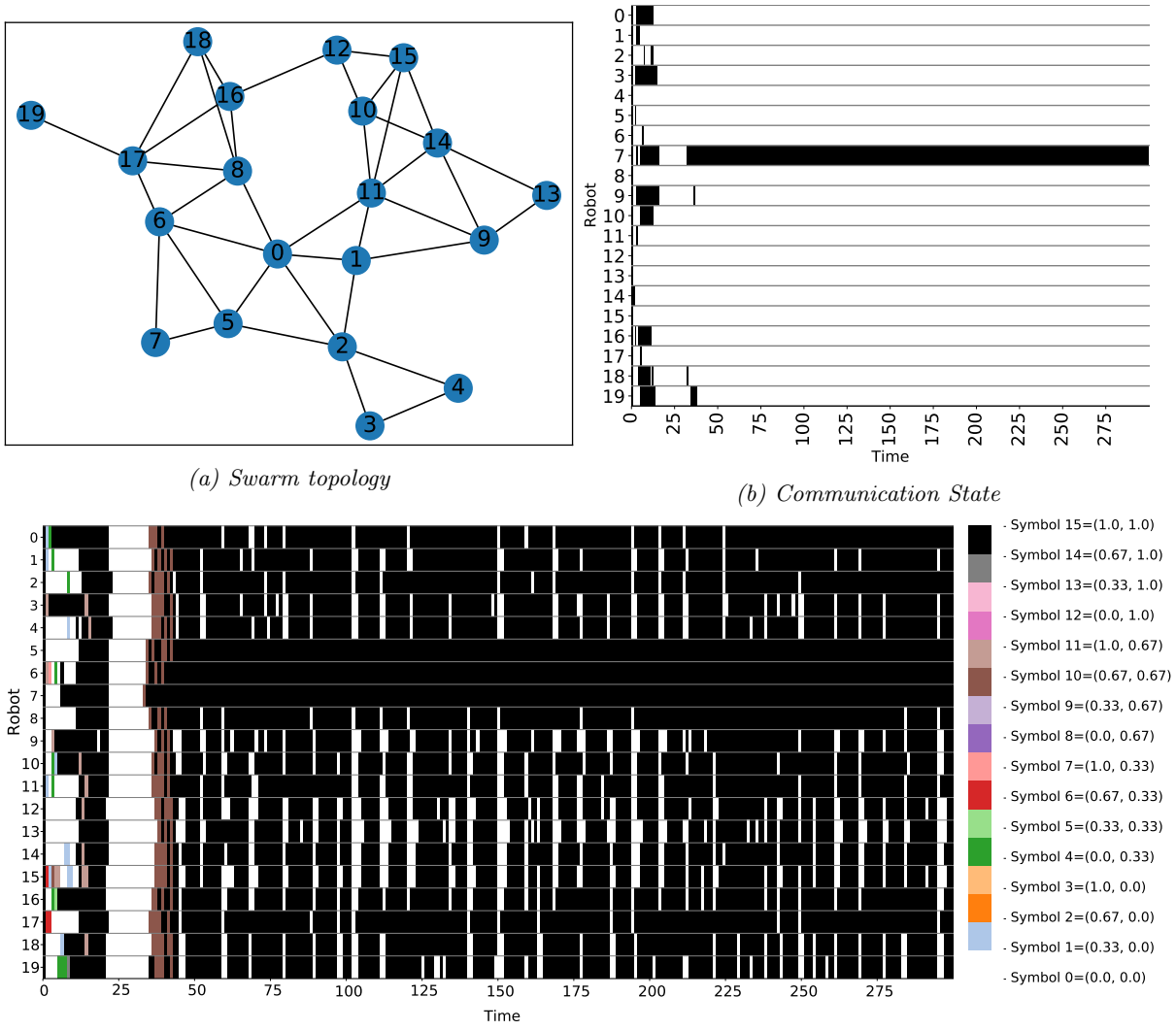


Figure 6.8: Percentage of simulation time with a single leader elected when communication stimulus are inhibited and in normal conditions. Stimulus inhibition is performed by replacing with zeros the corresponding input stimulus at neural level. Inhibition of the variables is performed separately and one by one.

Figure 6.8 shows the resulting performance of the solution in terms of percentage of time with

elected leader when variables are inhibited. Interestingly, the algorithm uses the message content, albeit the orientation from where the message was received seems to be irrelevant. This results are utterly important as they reflect that the message reception orientation could have been omitted in the architecture.



(a) Swarm topology

(b) Communication State

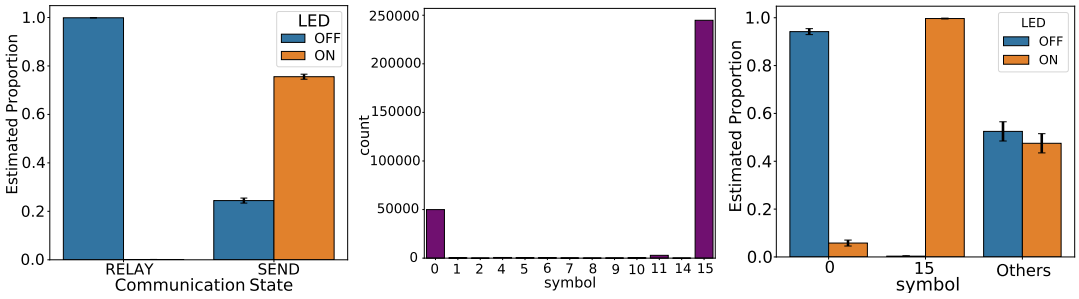
(c) Transmitted Message

Figure 6.9: (a) Spatial graph of the swarm topology, preserving positions and distances. (b) Communication state of each robot at each time step. Horizontal black bars correspond to state send and horizontal blank bars denote relay state. (c) Message transmitted by the robots at each time instant. The color of the bar at each time step corresponds to a symbol specified in the legend.

In the light of the previous observations, hereafter, attention is paid to the message vector content and to the communication state in order to understand the mechanics of the agents interactions. Figs. 6.9a shows the swarm spatial configuration, while Fig. 6.9b displays the communication state of each robot at each time step. Black bars indicate that the corresponding agent in the ordinate axis is in send mode and blank spaces mean relay mode of the robot. Note that there is a clear correlation between the communication mode of

the robots and their LED action. Specifically, most of the time that robots are in send mode, the LED claiming leadership is turned on (see Fig. 6.2). Additionally, Figs. 6.9c displays the message sent or relayed at each time step. The legend shows the possible symbols and the corresponding 2-dimensional vector. Provided that Figs. 6.9b and c are analyzed jointly, the following communication mechanics can be assumed.

- If an agent claims leadership, its communication state settles as send state. On the contrary, non leader robots enter into relay mode.
- Agents in send mode mostly emit symbol 15, corresponding to message $(1, 1)^\top$.
- The message sent by the potential leaders is spread around the entire swarm, resembling a wave-like propagation. In fact, Fig. 6.9a clarify that the agents sending symbol 15 more frequently are those with less number of hops to the leader (node 7).
- Messages between symbols 0 and 15 are essentially transient messages caused during the rise time of the motor neuron firing rate. This implies that only symbols 0 and 15 are actually relevant for the communication, and, therefore, there is no need to include more than 2 symbols in the communication of this experiment. More precisely, it is a signalling based communication.



(a) *(b)* *(c)*

Figure 6.10: (a) Estimation of the proportion of times of each LED status of robots conditioned to the communication state. (b) Count plot, gathering the times that each symbol is emitted by any robot in the entire dataset, formed by 50 independent episodes. (c) Estimation of the proportion of times of each LED status of robots conditioned to transmitted message. Symbols from 1 to 14 are merged into the "Others" category due to their minimal relevance. In (a) and (c), the point estimates are the upper sides of the bars and the confidence interval with a confidence level of 95% is showed as the black segment.

We reinforce the previous statements with an statistically significant sample of simulation trajectories. Fig. 6.10a shows a barplot depicting the estimated proportion of time that a robot that claims leadership conditioned to being in each of the communication states. The point estimates of the proportions, corresponding to the upper side of the bars, show that almost in the totality of times that a robot is in relay mode, it turns off the LED. On the contrary, the majority of robots in send mode have their LED activated. These results match with the observation in Fig. 6.9. In addition to the estimate shown as the top line of each bar, a confidence interval of the proportion estimate is also provided. The interval is computed as:

$$CI(p)_\alpha = \hat{p} \pm z_{\alpha/2} \sqrt{\frac{\hat{p}\,(1-\hat{p})}{n}} \tag{6.1}$$

where $\hat{p}$ is the proportion estimate, n is the sample size and $z_{\alpha/2}$ is the upper $\alpha/2$ percentage point of the standard normal distribution. The significance level α is set to 0.05, leading to

a confidence level of 95%. All the proportion estimation in the remaining of the chapter will use this significance level. Figure 6.10b gathers the number of times that each message is sent by any robot in the entire dataset of collected simulations. Clearly, the most frequent symbols, and thus the most meaningful ones, are symbols 0 and 15 (corresponding to $(0,0)^\top$ and $(1,1)^\top$, respectively). In the light of this fact, the remaining symbols are merged into the Others category in Fig. 6.10c in order to clarify the results. It shows the point estimates of the proportion of times of LED actions conditioned to the transmitted symbol. Only the symbols emitted when the communication is in send state are considered. It also depicts the confidence intervals with 95% of confidence level. The bars in the plot highly corroborate the previous assumptions stating that leader communicates its lead role to the swarm by sending symbol 15. Moreover, it also indicates that non leader robots mainly transmit symbol 0 and there is no statistical evidence suggesting that the rest of the symbols have an impact on the LED action.

6.2 Experiment B: Borderline Identification

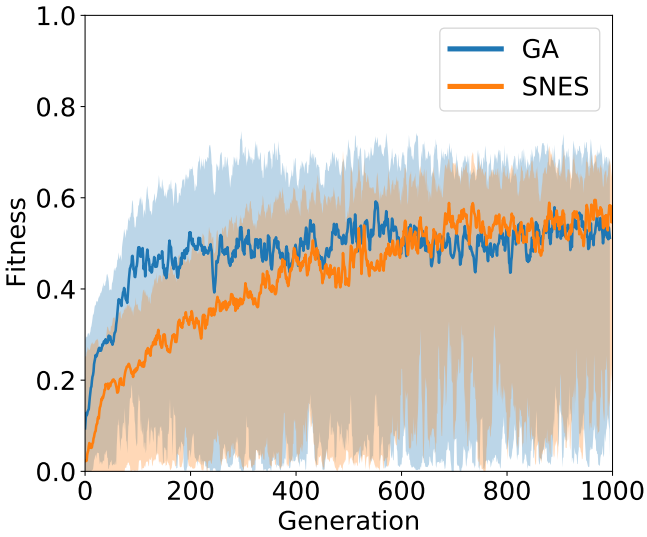
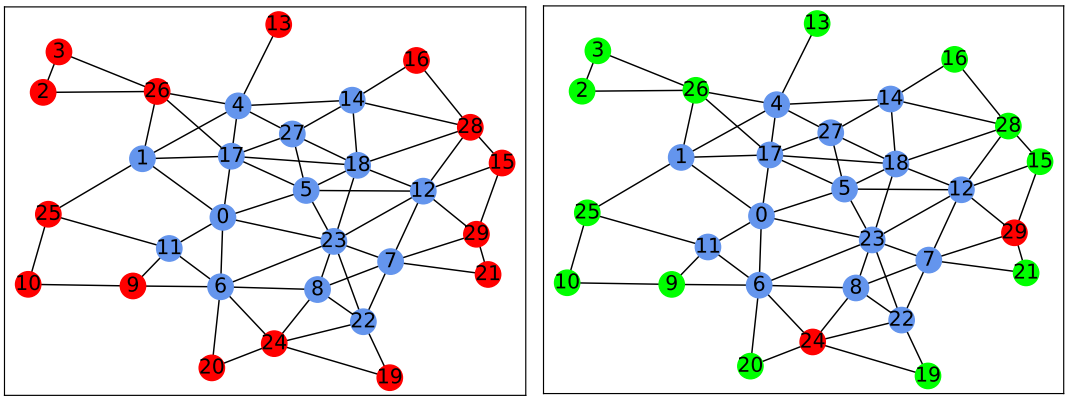


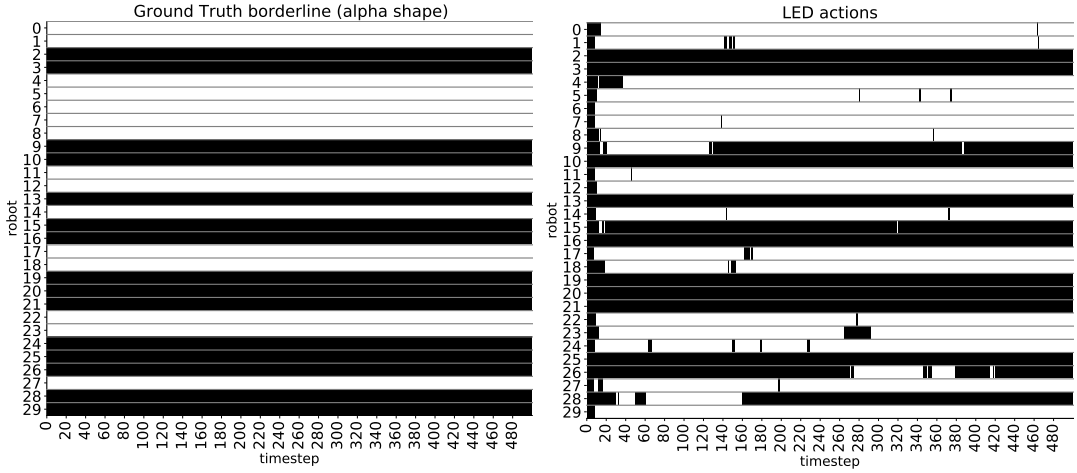
Figure 6.11: Evolution of the fitness function with the generations of GA and SNES in the border identification task. In each generation, the darker curves are the sample mean fitness scores and the upper and lower contours of the shadow areas represent the maximum and minimum fitness values.

Fig. 6.11 illustrates the evolution of the fitness value with the generations of both GA and SNES. It shows the sample mean fitness among all the genotypes of each generation as the darker curves. Moreover, the maximum and minimum fitness scores are additionally displayed as the contours of the shadow areas. It is worth mentioning that the high variability is not only caused by variability of the GA genotypes or the stochastic sampling of individuals in SNES but also because of the random episode initialization. It can be observed that both algorithms approximately converge to the same mean fitness value with a sufficient amount of generations. The main difference in the evolution is that, in this experiment, GA converges

in less generations than SNES. In terms of behavior and communication complexity, which is analyzed hereafter, the solutions of both algorithms present similar characteristics. We decided to focus on GA's solution because of its fast convergence, but the analysis results are also applicable to SNES.



(a) Swarm topology graph, alpha shape highlighted *(b) Swarm graph, successes and errors highlighted*



(c) Alpha shape *(d) LED actions*

Figure 6.12: (a) Spatial graph of the swarm, edges denote the existence of a pairwise communication channel. Red balls represent alpha shape members and blue dots are interior robots. (b) Spatial graph of the swarm, red balls denote agent errors (as indicated in (d)) and green balls denote correct borderline classifications. (c) Target borderline members according to the alpha shape. Horizontal black bars denote frontier robots and horizontal blank bars represent interior agents. (d) Temporal evolution of LED actions of the robots as time elapses. Horizontal black bars denote activated LED.

Behavior: In order to assess the correct functioning of the evolved solution, Fig 6.12 shows the behavior of the robots in a trial with 30 agents. Firstly, Fig 6.12a, illustrates the swarm spatial distribution and pairwise communication channels. Red balls represent robots belonging to the alpha shape with $\alpha = 15$ (see Section 5.2) and blue dots represent interior nodes. For the depicted graph, Fig 6.12c exposes the target LED actions or target alpha shape that the robots should perform in order to correctly identify the swarm borderline. Thus, the robots with horizontal black bars are members of the borderline according to the alpha shape algorithm. Additionally, Fig 6.12d shows the actual LED actions of the robots,

indicating if they are in the frontier or in the interior. Fig 6.12b displays, in the spatial swarm distribution, the correct frontier classifications (in green) and the errors (in red). The actions of Fig 6.12b correspond to a snapshot at time instant 480. The results are, in general, remarkably outstanding, albeit there are some robots whose decision is incorrect. Specifically, there are about 2 or 3 robots, depending on the observed time step, whose classification is wrong. It can be observed that there are some agents whose LED actions are remarkably stable (e.g. robots 2, 3 or 10) while the decisions of other robots are much more undefined (e.g. nodes 9, 26 or 28). A common observable feature is that agents with a robust decision generally have only few neighbors (1 or 2). On the contrary, unstable robots mainly belong to a dense part of the graph, with many neighbors (3, 4 or 5). Thereafter, the errors principally occur when the agents have many neighboring robots, leading to the naive classification of being interior node (which is an incorrect assumption in some cases).

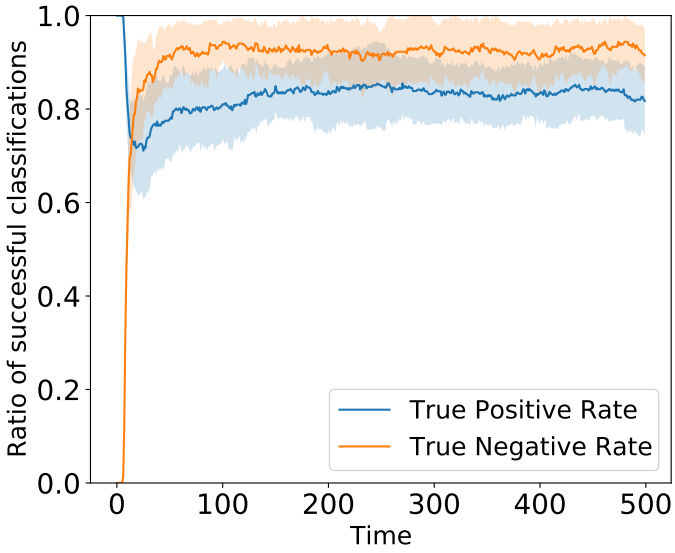
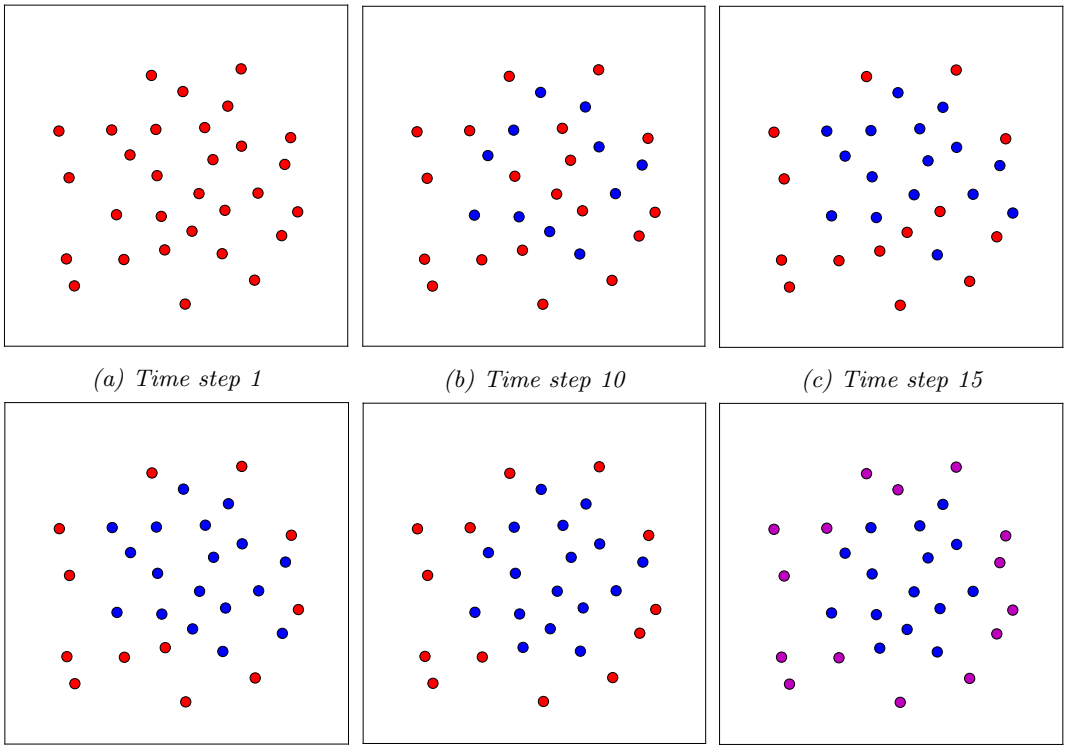


Figure 6.13: Temporal evolution of true positive and negative rates in the borderline identification experiment with 30 robots. Darker curves represent median TPR and TNR and contours of the shadow areas are the first and third quantiles, using a sample of size 50.

Furthermore, another interesting observation is that errors are more likely to appear as false positives as it can be observed in Fig. 6.13, where the true positive rate (TPR) is slightly inferior than the true negative rate (TNR). At the initial time steps of the simulation, the TPR and the TNR are respectively 1 and 0 because, at the simulation startup, all the agents identify themselves as frontier nodes (see Figure. 6.12d).

To observe the behavior in a more visual manner, Figure 6.14 collects snapshots of a sample trial at different time instants of the simulation. At the initial time step, all robots consider themselves as extremities (red balls). Subsequently, as time elapses, the solution is corrected until the final decision is settled (approximately at time instant 50). The actual alpha shape of the example is shown in purple in Figure 6.14f. The final classification of this example results in two errors, corresponding to false positives in both cases. Finally, the behavior of the solution to this experiment can be observed in video format [2].

[2] https://youtu.be/tGytXx2BM2w



(a) Time step 1 *(b) Time step 10* *(c) Time step 15*

(d) Time step 30 *(e) Time step 50* *(f) True alpha shape*

Figure 6.14: From (a) to (e), snapshots of the borderline identification experiment at different simulation time steps. The balls represent the robots in the swarm. Robots colored in red indicate that the LED is turned on at the corresponding time step. Similarly, blue balls denote robots with the LED deactivated. Swarm topology and robot distances are preserved in the graphs. (f) Actual alpha shape, in purple, used as target, Note that at time step 50, once the decisions are settled, there are only 2 errors. Moreover, both errors correspond to false positives.

Scalability: In Fig. 6.15, the scalability of the system is evaluated by The results are presented using a sample of size 50 with independent simulation executions. For each swarm size, it shows the accuracy of each time step, defined as the number of agents correctly identified as frontier or interior divided by the swarm size. The curves represent the time dependent median of the accuracy using all the collected samples. Alternatively, the shadow areas indicate, at each time instant, the first and third and quantiles of the accuracies. It can be observed that the solution scales utterly well as swarm size increases. Indeed, there is no statistically significant degradation in accuracy as the number of robot grows, up to 50 agents. Consequently, the scalability of the system is clearly fulfilled.

Robustness: The robustness capabilities are evaluated by reinitializing the robots spatial graph, defining their positions, every 200 time steps. However, the states of the CTRNNs of all the robots are maintained untouched regardless of the position resampling. This causes an abrupt change in the swarm topology, leading to agents being forced to reconsider if they are in the borderline subset. We decided to switch the topology with this procedure in order to avoid robot movement that would be of complex implementation and does not guarantee swarm compactness. Fig. 6.16 displays both target borderline temporal evolution (left) and robots LED actions (right). Note that the borderline switches every 200 time steps. Although there are several erroneous decisions, robots are generally able to respond successfully to topological

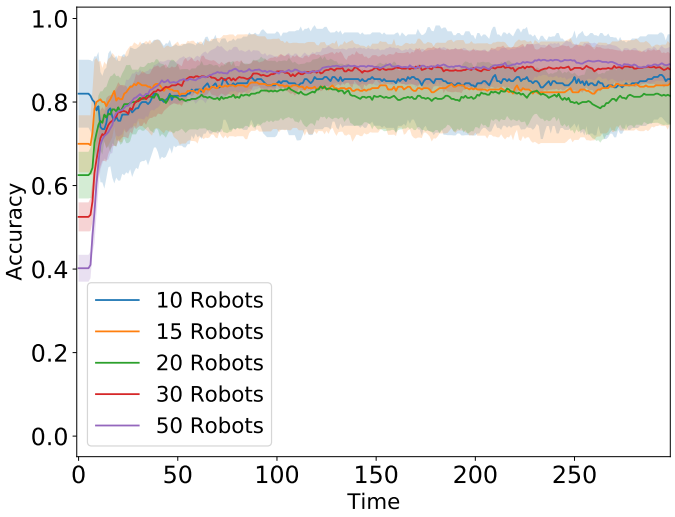


Figure 6.15: Temporal evolution of the accuracy of the robot's classification in the borderline identification experiment for diverse swarm sizes. The darker curves represent the median of the accuracy using all 50 collected samples. Alternatively, the shadow areas indicate, at each time instant, the first and third and quantiles of the accuracies.

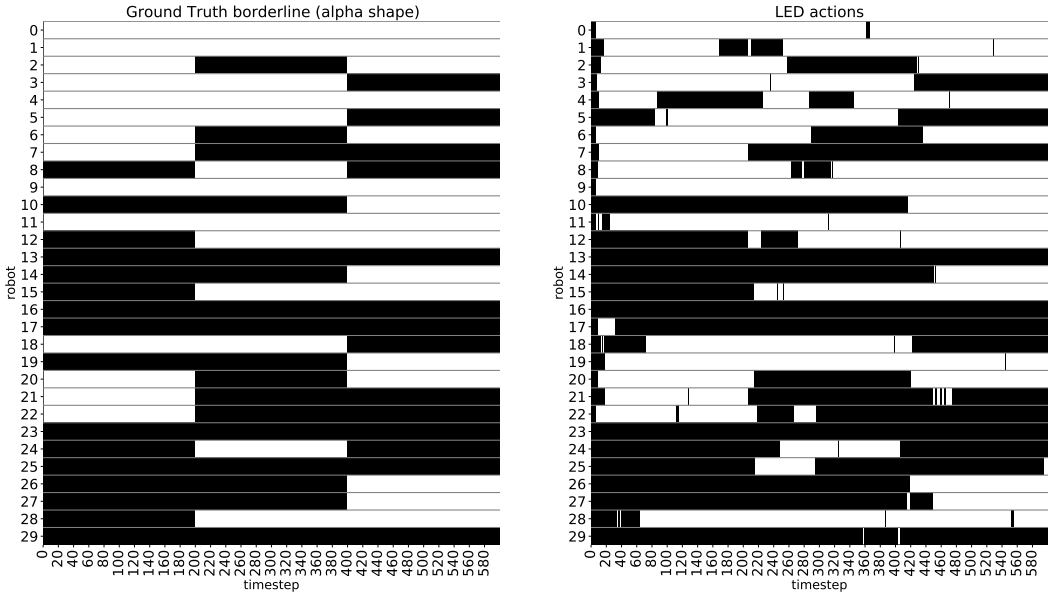


Figure 6.16: (Left) Target frontier members (according to alpha shape). Black bars indicate that the robot is in the alpha shape and blank bars represent interior nodes. (Right) Robots LED actions. The swarm topology is switched to a different one (randomly sampled) every 200 time steps.

alterations. Evidently, there is a short transient period of time required by the agents to notice the swarm changes and alter its neural states as a consequence.

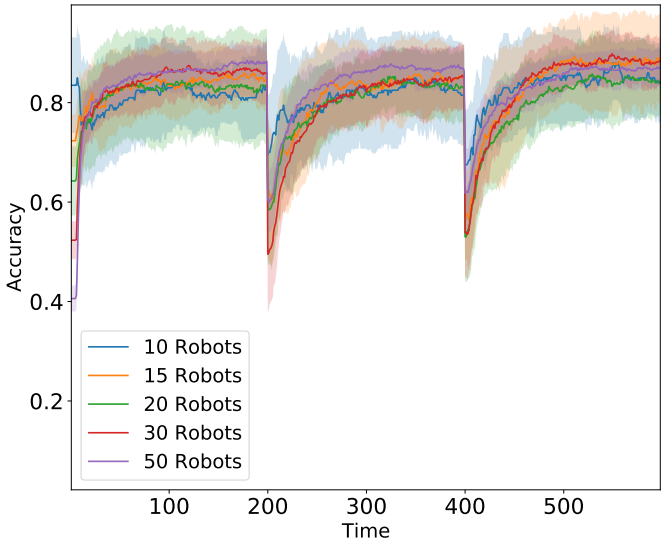


Figure 6.17: Temporal evolution of the accuracy of the robot's classification in the borderline identification experiment for diverse swarm sizes. Every 200 time steps the swarm topology is changes while the neural states are preserved. The darker curves represent the median of the accuracy using all 50 collected samples. Alternatively, the clearer areas indicate, at each time instant, the first and third and quantiles of the accuracies.

To complement the previous figures, Fig. 6.17 represents the accuracy distributions for different swarm sizes under the above mentioned topology alterations. As in Figure 6.15, the shadow area indicates the first and third quantiles of the accuracy at each time instant, while the darker curves highlight the median. The resampling of robot positions result in discontinuous accuracy drop as the alpha shape and, thus, the robots in the borderline, change. Agents are able to detect the changes and almost correctly solve the task for the unexpected swarm redistribution. Exactly as in Fig 6.15, the accuracy is equivalent as swarm size grows. Moreover, the transient reaction time after topology change is remarkably similar regardless of the number of robots.

Communication: With the aim of studying the communication that emerges as a result of evolution, Fig. 6.18 analyzes the importance of the communication variables in solving the task. Specifically, in the figure, it can be observed the accuracy comparison among a scenario with no inhibition and when different communication variables are inhibited. Each inhibition is performed one by one and consists of replacing the value of the corresponding variable with zeros at the input of the CTRNN. The comparison of the figure reveals that the information about the orientation from where the message was sensed is a highly relevant context variable, whose deletion causes system breakdown. On the contrary, provided that the system accuracy is not significantly decreased, the results indicate that the message is not harnessed in this experiments. This leads to a solution to the task with purely situational communication, in which only the context underlying the message carries information. The communication state was not included because we observed that the CTRNN of the robots always settle the state to send mode. In relation to the message reception orientation, it is remarkably challenging to find out the precise functioning and usage of its information. However, Fig. 6.19 illustrates the estimate of the proportion of the time that robots have their LED turned on conditioned to the number of neighbors. The plot equally shows the confidence interval with a confidence level of 95% (see Eq. 6.1). It can be directly observed that when an agent has four neighbors,

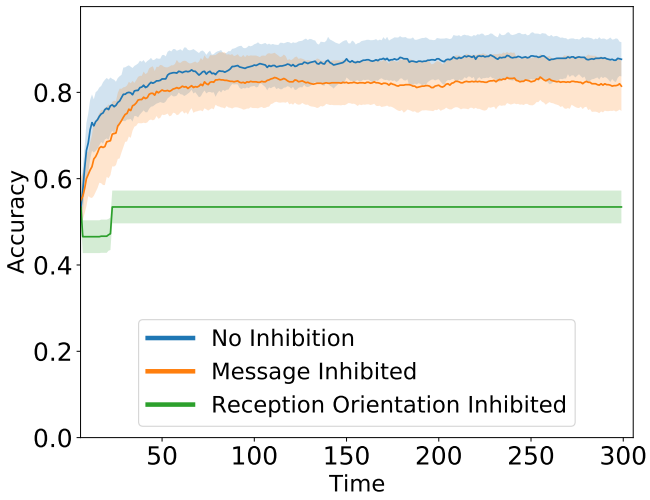


Figure 6.18: Temporal evolution of the accuracy of the robot's classification in the borderline identification experiment for different inhibited variables. It compares the accuracy in a situation without inhibition (blue) and inhibiting different communication variables (one by one).

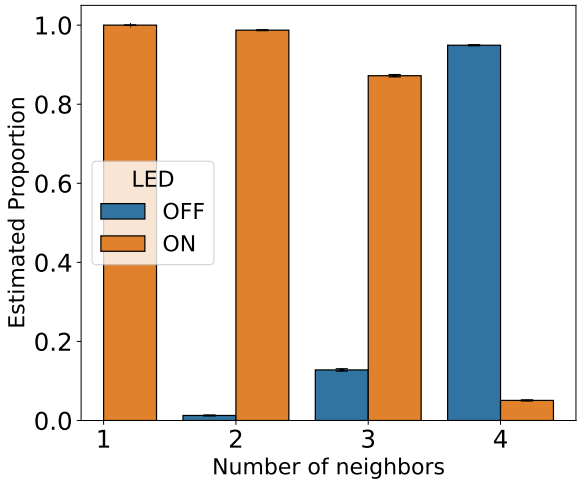


Figure 6.19: Estimation of the proportion of times that the LED is activated, and thus robot classifies itself as borderline member, conditioned to the number of neighbors sending messages from different orientations. The upper side of the bar indicate the proportion point estimate with its corresponding 95% confidence interval.

it identifies itself as an interior member in the absolute majority of times. Alternatively, when it has one or two neighboring robots, it clearly assumes its membership to the borderline of the swarm. In the case of three neighbors, the agent also decides to turn the LED in most of the cases, albeit the number of LED deactivations are increased with respect to the situation with two neighbors. It is worth mentioning that the number of neighbors as stated in Fig. 6.19 is not computed directly based on distances between robots. Alternatively, it is obtained using the number of messages that a robot receives from different orientations within a time window. Moreover, it must be kept in mind the noticeable fact that only one message is received at each simulation cycle. This means that, in order to gain insight into the number of neighboring robots, the CTRNN of the agent must retain in memory information about the message context at previous time steps.

6.3 Experiment C: Orientation consensus

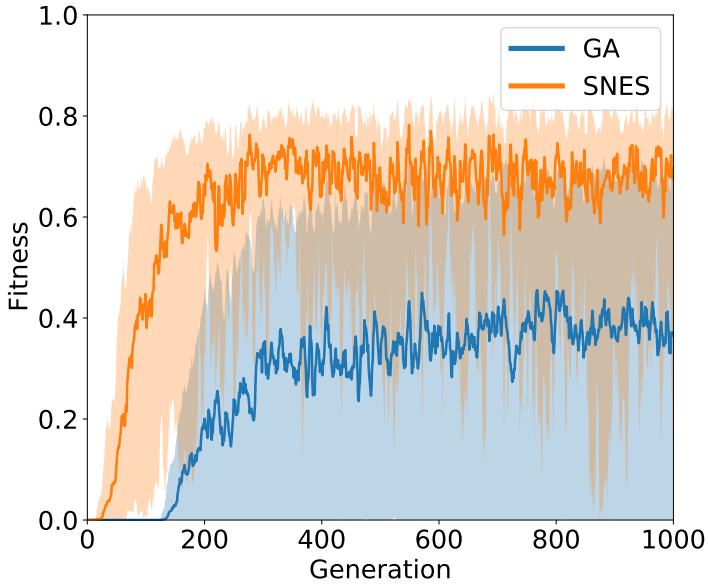


Figure 6.20: Evolution of the fitness function with the generations of GA and SNES in the orientation consensus task. In each generation, the darker curves are the sample mean fitness scores and the upper and lower contours of the shadow areas represent the maximum and minimum fitness values.

A comparison of the performance, in terms of achieved fitness scores, between GA and SNES is provided in Fig. 6.20. It displays how the fitness of the individuals evolves as generations elapse. Darker curves indicate the sample mean fitness values and the upper and lower limits of the shadow areas indicate the maximum and minimum fitness scores in each generation. The genotypes reached by SNES are clearly superior in terms convergence speed and steady state fitness achieved once convergence is reached. Moreover, SNES starts to increase the fitness of its individuals at early generations, while GA elapses about 170 generations with nearly zero fitness solutions. For all these reasons, we focus on the solution provided by SNES for the detailed behavioral and communication analysis.

Behavior: In the light of the fitness function results previously shown, the assessment of the behavior is presented. As it is exposed below, the evolved agents successfully solve the task of orientation consensus under the minimal communication capabilities at their disposal. Fig. 6.21 displays an example of the results with a simulation with 20 robots. Each curve represents the instantaneous orientation in radians of the robots. We do not include a legend clarifying which curve corresponds to which agent because it is not actually relevant for the figure interpretation and would ruin the graph due to the large swarm size. Robots are randomly initialized at diverging orientations. After a transient period of about 100 time steps, the robots tend to reach the orientation consensus by matching their heading direction with the orientation of its neighborhood. It should be mentioned that, even though consensus is approximately fulfilled, robots still rotate with very low angular speed in order to preserve orientation agreement. This residual rotation can be observed in the figure as the slope in the orientations of the robots, albeit we remark that the slope is merely about 0.01 radians

per time step.

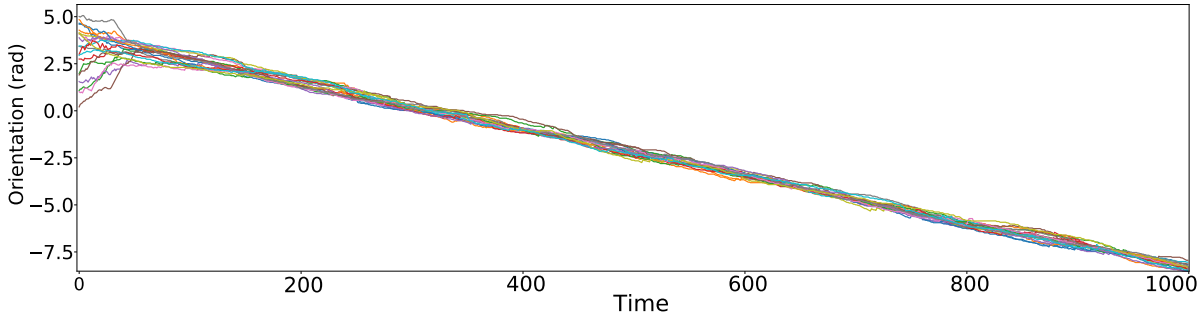
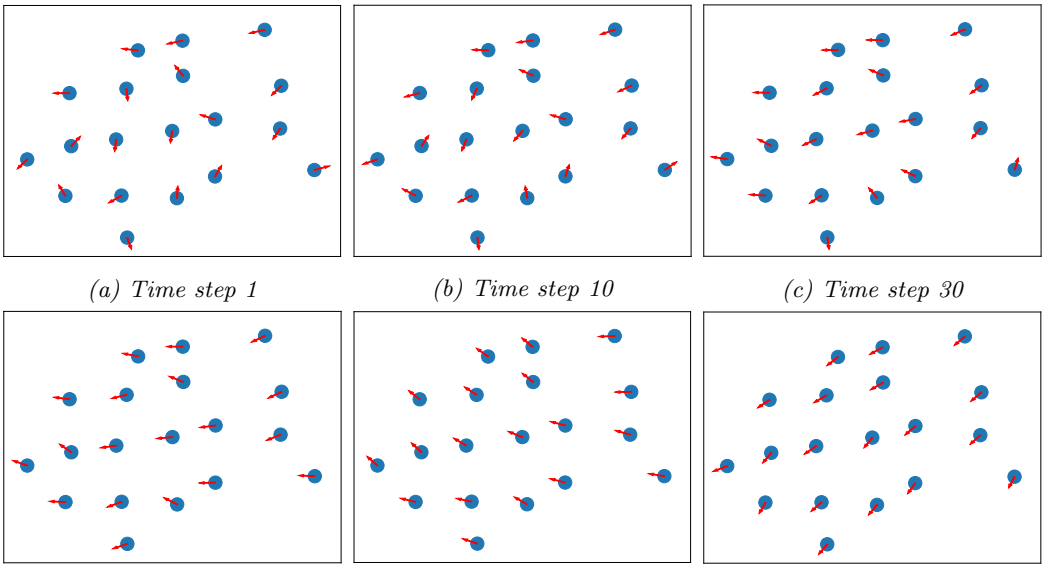


Figure 6.21: Temporal evolution of the orientation of the robots in a simulation with swarm size of 20. Each curve corresponds to the orientation of one of the agents. The orientation range of $[0, 2\pi]$ is extended to the $\mathbb{R}$ set merely for visualization purposes.



(a) Time step 1 *(b) Time step 10* *(c) Time step 30*

(d) Time step 50 *(e) Time step 100* *(f) Time step 500*

Figure 6.22: Snapshots of different time instants in a simulation of the orientation consensus experiment. Blue dots depict the robots in the swarm and red arrows show the orientations of the agents.

The goodness of the collective behavior can be equivalently observed in Fig 6.22, where snapshots of the orientations of the robots at different time steps of the simulation are plotted. Blue balls represent the robots in the swarm and red segments illustrate the heading orientation of the robots. Finally, the behavior of the solution to this experiment can be observed in video format [3].

Scalability: We now assess the scalability properties of the collective behavior. Moreover, as in pervious experiments, the results exposed in Fig. 6.21, that uniquely represent one sample that could be biased, are reinforced by collecting and using 50 trial simulations for constructing reliable assessments.

[3] https://youtu.be/bTY2x9Aw9s4

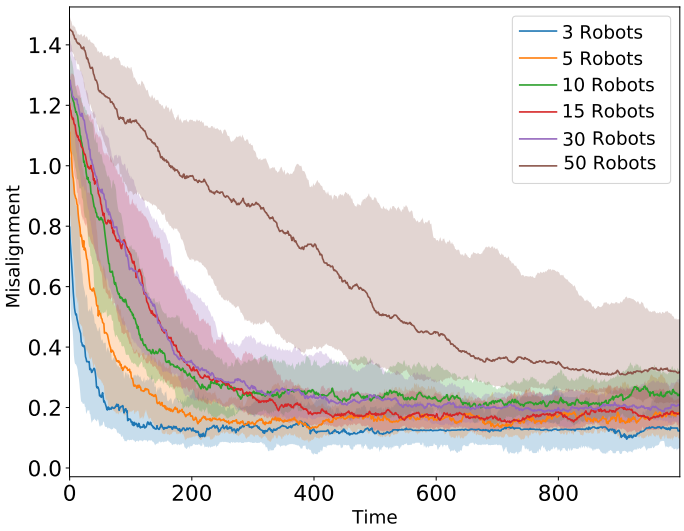


Figure 6.23: Temporal evolution of the misalignment metric (see Eq. 6.2) distribution using 50 simulation trials and diverse swarm sizes. The darker curves represent the median of the misalignment using all 50 collected samples. Alternatively, the clearer areas indicate, at each time instant, the first and third and quantiles.

Thereafter, Fig. 6.23 visualizes the performance with diverse swarm sizes and using the 50 samples. Before commenting the figure, we introduce the misalignment metric defined as in Eq. 6.2. Note that it is included in one of the terms of the fitness function in Eq. 5.5, albeit it is now formally introduced as a metric for performance assessment.

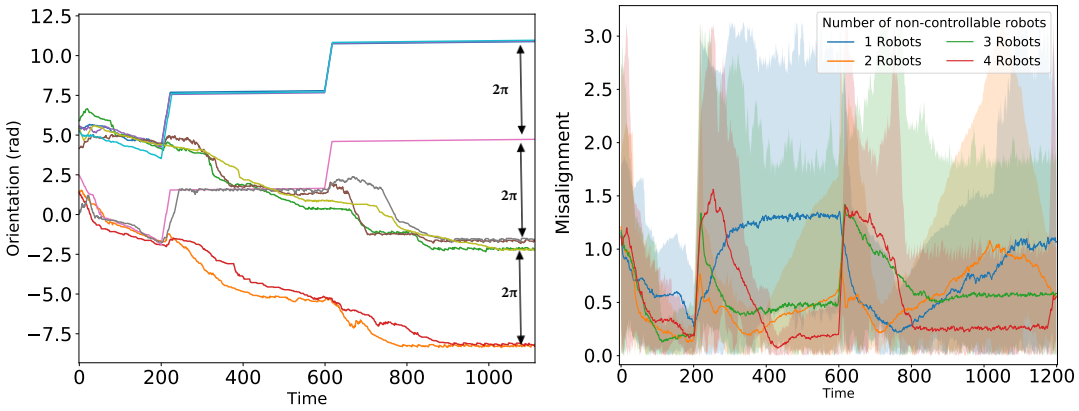
$$M_\theta(t) = \frac{1}{R}\sum_{r\in\mathcal{R}} \min\left\{|\theta_r(t) - \overline{\theta}(t)|, 2\pi - |\theta_r(t) - \overline{\theta}(t)|\right\} \tag{6.2}$$

It essentially measures the mean orientation deviation of each robot with respect to the mean orientation of all robots. Similarly, the mean orientation $\overline{\theta}$ is computed as:

$$\overline{\theta}(t) = \frac{1}{R}\sum_{r\in\mathcal{R}} \min\left\{\theta_r(t), 2\pi - \theta_r(t)\right\}$$

Once the misalignment metric is defined, the interpretation of Figure 6.23 is resumed. At each time instant, the darker curves denote the sample median of the misalignments of the 50 trials. Moreover, the shadow areas are delimited by the first and third sample quantiles of the misalignment metric. In contrast to the border identification experiment, the results show a clear degradation of the misalignment as the swarm size increases. The worsen is noticeable in both transient time required to reach orientation consensus and in steady state misalignment. Nonetheless, the degradation starts to be noticeable somewhere in between swarm sizes of 30 and 50. Thus, it can be stated, that although not perfect, the scalability properly is correctly fulfilled.

Robustness: The unexpected perturbation introduced during the evaluation simulation, with the aim of testing the robustness, is the following. At time step 200, a subset of the robots in the swarm becomes "uncontrollable" in the sense that their wheel actuator is not managed by the neural controller anymore. More precisely, the robots in this state rotate at constant angular velocity until they reach a precise orientation. All these agents tend to the same orientation, arbitrarily chosen by us. Moreover, at time step 600, the objective of the "uncontrollable" robots changes to a different fixed direction, so that these agents start to



(a) *(b)*

Figure 6.24: (a) Temporal evolution of the orientation of the robots in a simulation with swarm size of 10 for the robustness assessment. Each curve corresponds to the orientation of one of the agents. At time instant 200, 4 robots become "uncontrollable" and point to the same orientation. The rest of the robots are controlled by the neural controller. At time step 600, the "uncontrollable" agents change their orientation. Notice that the y-axis values with a difference of 2π correspond to the same physical robot orientation. (b) Temporal evolution of the misalignment metric (see Eq. 6.2) distribution using 50 simulation trials under the conditions specified for (a). The simulation misalignment is tested for different quantities of "uncontrollable" robots. The darker curves represent the median of the misalignment using all 50 collected samples. Alternatively, the clearer areas indicate, at each time instant, the first and third and quantiles.

rotate again until they reach the new goal. Fig 6.24a shows the orientation of all the robots with the mentioned mechanic included. There are 4 robots that become "uncontrollable", corresponding to the curves with constant orientation in the previously specified time intervals. Notice that these curves start to vary for about 50 time instants at constant slope, until they reach the corresponding reference. Before we interpret the figure, it is important to clarify that the y axis distances highlighted by arrows are of 2π radians and, therefore, the curves with this separation correspond to the same physical orientation. In Fig 6.24a it can be seen how robots that are not affected by the imposed constrains are capable of reacting to the drastic orientation switch of the leaders. Furthermore, this modification outstandingly improves the solution as agents tend to maintain their heading orientation more robustly than in the vanilla experiment (see Fig. 6.21). For instance, in the last 200 time steps, it is clearly visible that the robots correctly preserve their orientation, facing towards the direction imposed by the "uncontrollable" agents. From a general perspective, Fig 6.24b displays the misalignment metric using 50 simulation trials and varying the number of robots that belong to the "uncontrollable" subset of agents. Firstly, note that there is a misalignment increase at time steps 200 and 600, due to the goal direction introduction and alteration, respectively. Interestingly, as the number of robots subject to the new mechanics increase, the misalignment metric improves. In fact, 4 uncontrollable robots generally results in the lowest misalignment, albeit the time needed to reach steady state is longer. The reason of this observation is that a small number of robots changing their orientation are not enough to change the group orientation consensus. Apart from serving as a robustness evaluation method, the implemented alteration tries to simulate a scenario in which some leading robots are aware of a goal location and rotate until their headings point to the objective. In this way, they indirectly communicate the location of some objective with its orientation. Then, the rest of the robots mimic the heading direction of the leaders in order to reach orientation

consensus. Alternatively, this condition is also suitable for accomplishing one of the principal pillars of flocking, in which several robots decide the movement direction of the flock.

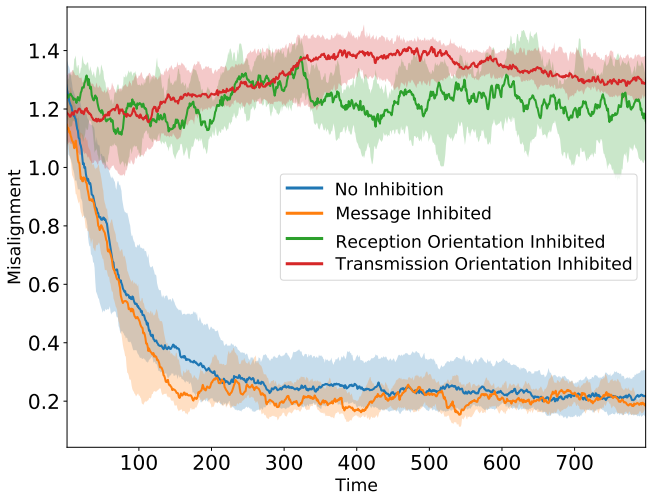


Figure 6.25: Temporal evolution of the misalignment metric (see Eq. 6.2) distribution using 50 simulation trials and for different inhibited communication inforamtion. The darker curves represent the median of the misalignment using all 50 collected samples. Alternatively, the clearer areas indicate, at each time instant, the first and third quantiles.

Communication: We now analyze the emerged communication mechanics and the communication information that is relevant for solving the task. Firstly, Fig. 6.25 shows the misalignment evolution when different variables are suppressed. As in previous figures, the darker curves denote the median sample estimate and the extremities of the shadows indicate the first and third quantiles. The deletion of the message, the transmission orientation and the reception orientation are considered by replacing by zeros the variable content at the CTRNN input level. The state of the communication was not studied because we observed that all the robots are always in the send mode. The curves in the figure indicate that the reception and transmission orientations are both crucial for solving the problem. On the contrary, it can be visualized that the message suppression leads to an equivalent misalignment evolution compared to the normal conditions. Therefore, apparently, this fact suggests that the message content by itself is not relevant for the experiment.

However, Fig. 6.26 provides a different perspective that refutes the previous statement. It compares the orientation temporal evolution for a trial. Black curves represent the orientations of the robots in the trial with normal conditions and, alternatively, red curves depict simulations with message content deleted. Clearly, the message content is used by the optimized CTRNN for drastically reducing the angular velocity of rotation while preserving the alignment with other agents (compare the slopes of the curve bundles). This property is not reflected in the misalignment metric and, thus, Fig. 6.26 incorrectly shows the message usefulness. To summarize, while the message reception and transmission orientations are principally used to fulfill the orientation consensus, the message content is complementarily harnessed for reducing the robots velocity, once aligned.

The violin plots displayed in Fig. 6.27 illustrate how the reception and transmission orientations are used to reach alignment. A violin plot is a descriptive statistics tool mainly used for visualizing the kernel density plots of continuous variables conditioned to the classes of categorical variables. In this case, each violin plot shows the estimated PDF of the wheel

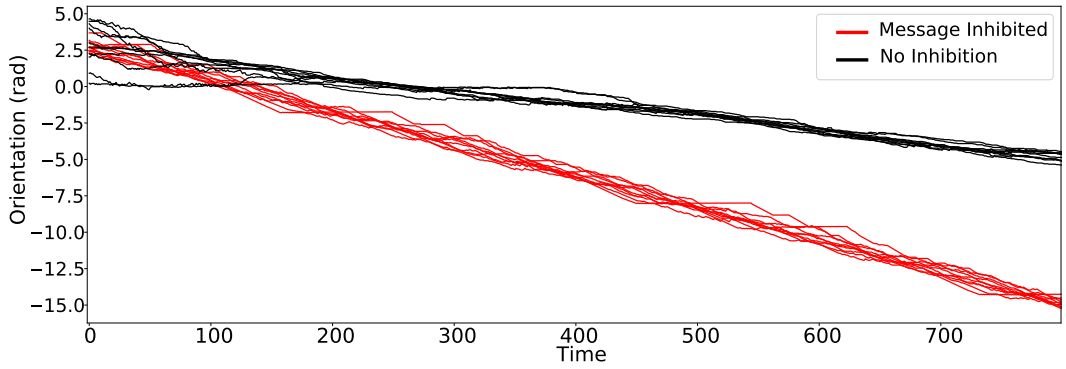


Figure 6.26: Temporal evolution of the orientation of the robots in a simulation with any communication variable inhibited (black) and with the message content inhibited (red). Curves in each color represent the orientations of the robots in the swarm in the corresponding simulation conditions. The orientation range of $[0, 2\pi]$ is extended to the $\mathbb{R}$ set merely for visualization purposes.

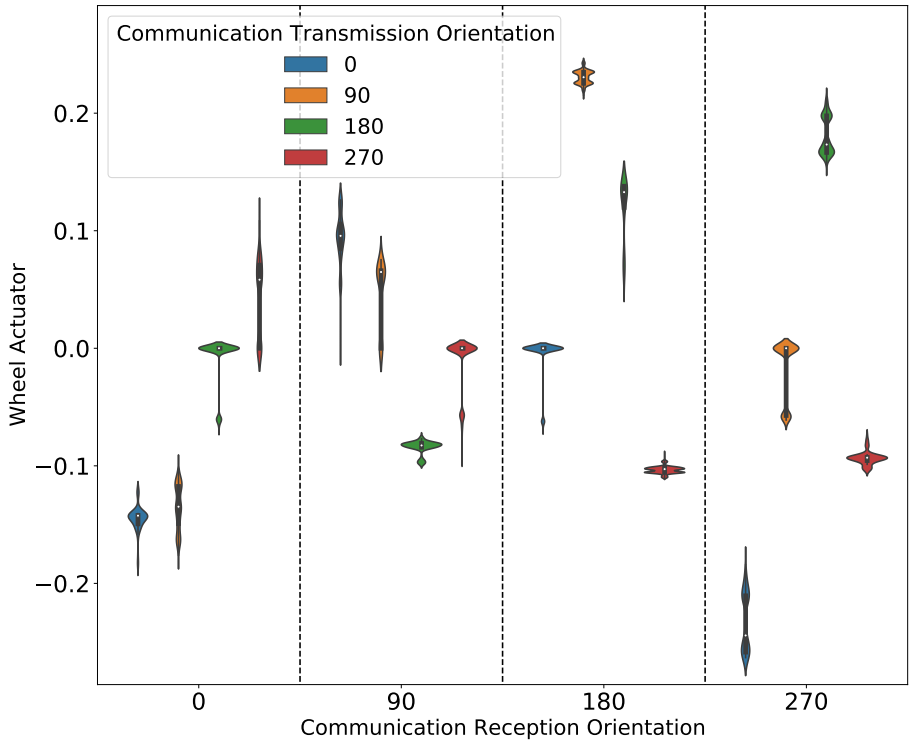


Figure 6.27: Violin plot of the wheel actuator (rotation) conditioned to the communication transmission orientation (θ_{TX}) and the communication reception orientation (θ_{RX}). A violin plot represents the kernel density estimation of each conditional distrubution.

action generated by the CTRNN conditioned by the orientations from where the input message was received and sent. All the conditioned distributions have enough samples to construct reliable kernel density estimates. Firstly, it is important to remark that an agent is aligned with the robot that transmitted the input message when the following condition is met:

$$|\theta_{TX} - \theta_{RX}| = 180 \tag{6.3}$$

Note that all the violin plots that fulfill this condition have a similar kernel density estimate,

whose samples are mainly gathered around 0 and -0.05. This observation indicates that when the robot is aligned, its wheel action tends to be zero (and thus the robot stops its rotation). In the other cases, the action is positive or negative, meaning clockwise or counterclockwise rotation, depending on whether the difference $|\theta_{TX} - \theta_{RX}|$ is lower or greater than 180. For instance, provided that the reception orientation is 270°, the action is positive when the transmission orientation is 180° and negative when it is 0°.

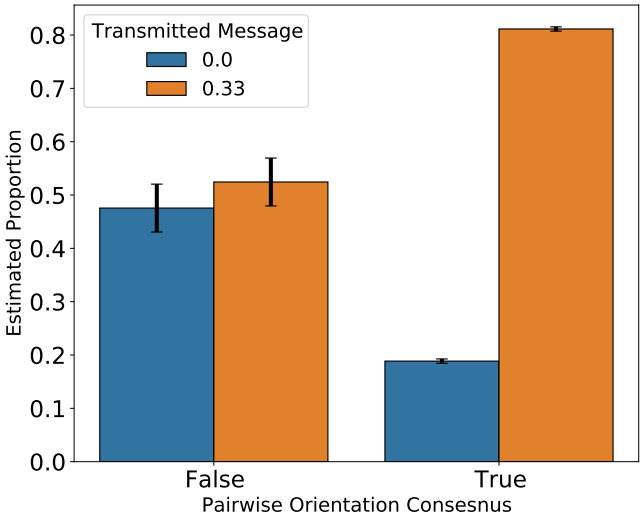


Figure 6.28: Proportion estimates and 95 % confidence intervals of the times each symbol is transmitted conditioned to the status of pairwise communication. Pairwise communication indicates if the sender and the receiver agents fulfill condition 6.3.

Regarding the exact role of the message content in the communication, Fig. 6.28 exposes the estimate of the proportion of times that a robot transits each symbol message when pairwise alignment is fulfilled. The CTRNN only generates symbols $(0, 0)^\top$ and $(0.33, 0.33)^\top$ and, thus, only those symbols are depicted in the figure. In addition to the point estimates, the plot additionally illustrates the confidence intervals with 95% of confidence level (see 6.1). By pairwise orientation consensus or alignment we refer to the situation in which, for a time instant, the sender and receiver agents fulfill the condition 6.3. It can be observed that when pairwise orientation consensus is reached, the symbol $(0.33, 0.33)^\top$ is mostly generated. Alternatively, when robots are not aligned, there is not statistically significant difference in the proportion of times that each symbol is generated. This information matches with the observations of Fig. 6.26, indicating the relevance of the message content once orientation consensus is achieved. Nonetheless, a finer and more detailed statistical analysis of the precise communication mechanics is left as future work.

6.4 Experiment D: Light follower

The generational evolution of the achieved fitness in the light follower experiment is presented in Fig. 6.29. As in previous sections, it shows the mean, maximum and minimum fitness values in each generation for both GA and SNES. Clearly, SNES outperforms GA with a convergence average fitness of 0.7. Therefore, in the behavioral and communication analysis, the solution provided by SNES is addressed in detail.

Behavior: The emerged behavior that is observed in the experiment of light or objective follower correctly solves the task. However, it involves several issues that are described

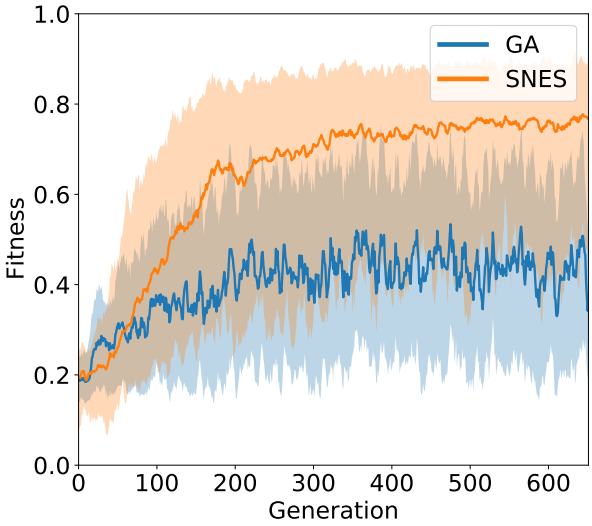
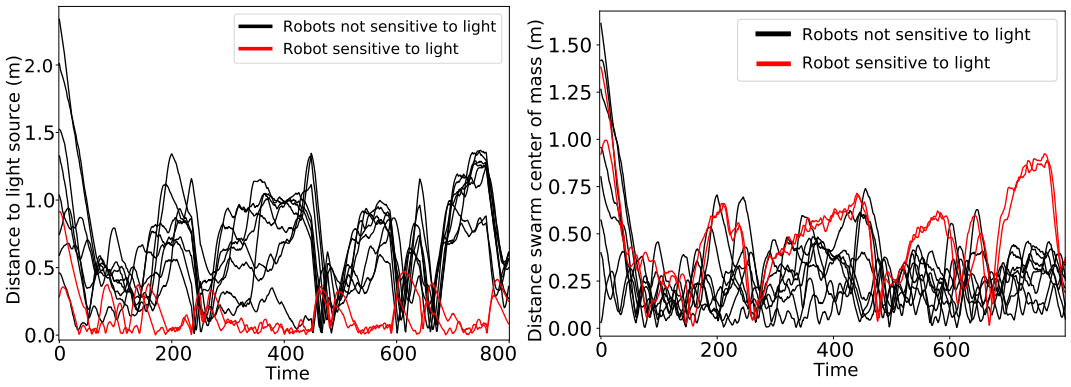


Figure 6.29: Evolution of the fitness function with the generations of GA and SNES in the light follower task. In each generation, the darker curves are the sample mean fitness scores and the upper and lower contours of the shadow areas represent the maximum and minimum fitness values.

subsequently. If the small subset of robots that can sense the light enter into the light source range, they instantaneously approach and reach the goal position. Moreover, these agents maintain a highly reliable tracking of the light once they capture it. The main issue is that the behavior of seeking the light source when they do not sense it has not emerged at all. A highly probable reason is that any agent measuring zero light intensity throughout its sensors considers itself as a non photosensitive robot, albeit it may not be the case. Alternatively, the observed behavior of agents whose light sensor is disabled is that they tend to cluster with neighboring agents. Therefore, a behavior of agent aggregation, which is also an interesting problem, has emerged from a light follower experiment. Agents that cannot sense the light source tend to the position of their potentially photosensitive neighbors, that could act as guides to the light source. Fig. 6.30 exemplifies this behavior by means of plotting the Torus

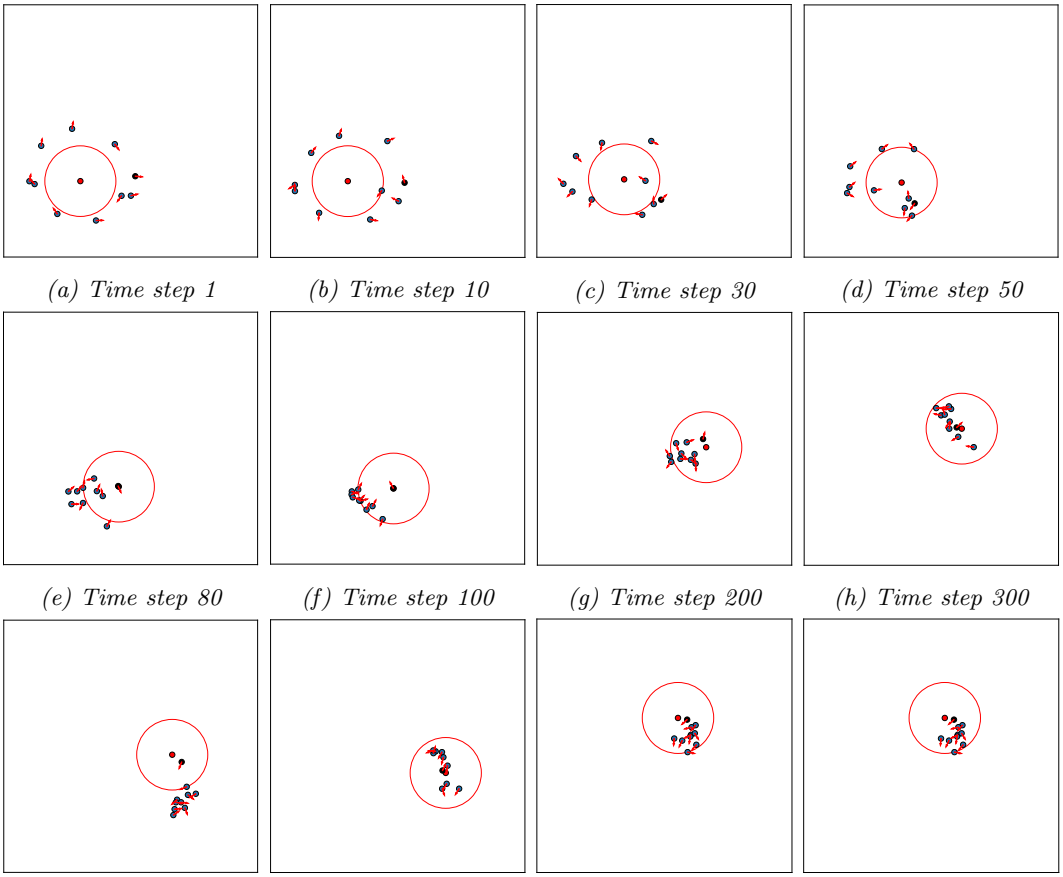


(a) Distances of robots to light source position

(b) Distances of robots to swarm center of mass

Figure 6.30: Torus distance $d_{\mathcal{T}}$ of each robot in the swarm to the light source position in (a) and to the center of mass of the swarm, in (b).

distance $d_{\mathcal{T}}$ of each robot to the light source position, in (a), and to the center of mass of the swarm, in (b). We distinguish between photosensitive (red) and non photosensitive (black) robots. In this case, there are 10 robots, from which only 2 of them can sense the light. In Fig. 6.30a, photosensitive robots' distance to the light is utterly low, albeit there are short periods of time when they momentarily move away from light. Nonetheless, these periods, that also correspond to the rest of robots approaching to the light source, happen because the light source reverts its movement direction (recall from Section 5.4 that the light source describes a circular trajectory whose rotation sense can be inverted with small probability). The sense inversion results in a drastic rotation (of 180 degrees) of the robots in order to follow the light source in the opposite direction. Regarding the non-photosensitive robots, their distance to the light is much larger, being generally about 1m, at most (note that 0.8m is the sensing range of the light sensor in this experiment). Alternatively, Fig. 6.30b displays the distances to the center of mass of the swarm. Opposite to Fig. 6.30a, this plot shows that photosensitive robots are the farthest agents to the center of the group. Essentially, robots represented by red curves closely follow the target light while the rest of the swarm group together and follow the photosensitive agents from the distance. Although



(a) Time step 1 *(b) Time step 10* *(c) Time step 30* *(d) Time step 50*

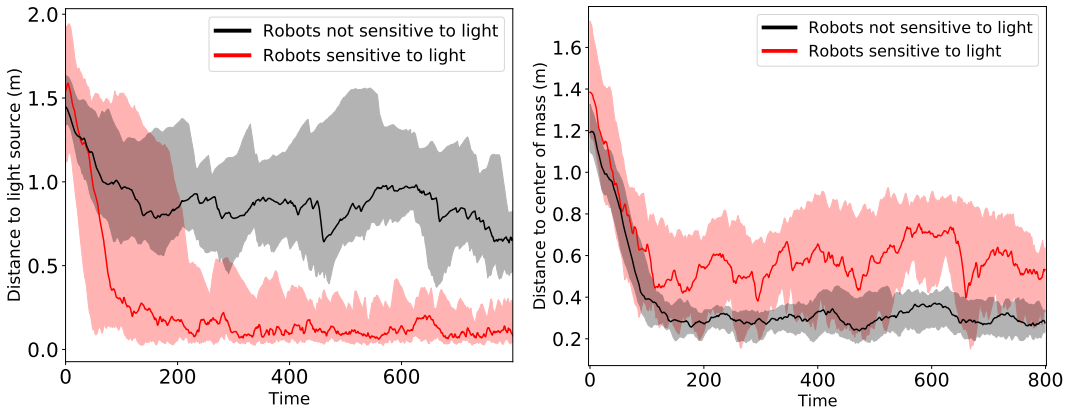
(e) Time step 80 *(f) Time step 100* *(g) Time step 200* *(h) Time step 300*

(i) Time step 500 *(j) Time step 600* *(k) Time step 700* *(l) Time step 800*

Figure 6.31: Snapshots of different time instants in a simulation of the light follower experiment. Blue dots depict the robots in the swarm and red arrows show the orientations of the agents. The red ball the is the light source, whose coverage or area where a robot can sense its emitted light is delimited by a red circumference.

it is not depicted in the figure, it is also important to mention that there are trials in which

the aggregated agents lose communication with the photosensitive robots, resulting in the breakup of the swarm compactness. Figure 6.31 shows frames of a simulation trajectory in this experiment. Blue balls denote robots with light sensor disabled and black ball represents the photosensitive agent (in this trial there is a single robot of this kind). Red segments illustrate robot's orientation and the red ball depicts the moving light source. Additionally, the red circumference delimits the farthest position from where light can be measured by any agent. As in previous experiments, a significant sample of simulation trials is collected.



(a) Distances of robots to light source position *(b) Distances of robots to swarm center of mass*

Figure 6.32: Torus distance $\mathrm{d}_{\mathcal{T}}$ distribution of robots to the light source position in (a) and to the center of mass of the swarm, in (b), using 50 trials. Darker curves represent the median evolution and the contours of the shadows are the first and third quantiles. In both subfigures, the black distribution encompasses the robots that cannot sense the light, while the red distribution corresponds to photosensitive robots.

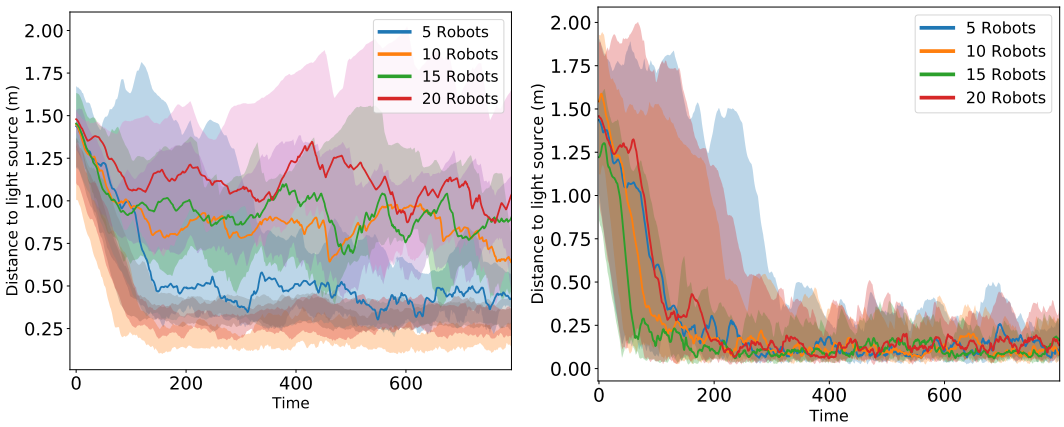
Fig. 6.32 represents the sampled distribution of the mean distance to the light source (a) and to the center of mass of the swarm (b) of the robots. The darker lines illustrate the sample median values and the shadow areas' contours show the first and third quantiles. Finally, the behavior of the solution to this experiment can be observed in video format [4].

Scalability: The number of robots is varied in order to assess the scalability of the solution. Fig. 6.33 gathers the results for 5, 10, 15 and 20 swarm sizes. In this case, Fig. 6.33a illustrates the distribution of the mean distance of the agents with light sensor disabled to the light source and Fig. 6.33b shows the distribution of the mean distance of photosensitive robots. Again, dark curves and shadow areas expose the median values and quantiles. It can be observed that the performance worsens in the case of non-photosensitive robots as swarm size increases. This degradation is principally caused by the fact that robots tend to aggregate and ignore photosensitive agents due to the large swarm size. In the case of agents sensitive to light, there is no evident degradation as the number of robots increase.

Robustness: In order to verify the robustness capabilities, we consider the following test. The trajectory followed by the light, used during the evolution phase, is replaced by a different one. The motivation of this modification is to validate that the light follower agents' behavior is not overfitted to the exposed light movement patterns. The novel trajectory described by the light is as follows. Every 50 time steps, a target position $\mathbf{x}_{tar}$ is randomly sampled within the torus. Thereafter, the position of the light is updated as,

$$\mathbf{x}_l(k+1) = \mathbf{x}_l(k) + \alpha\left(\mathbf{x}_{tar}(k) - \mathbf{x}_l(k)\right) \tag{6.4}$$

[4] https://youtu.be/HXLStH5za6Q



(a) Distances of non-photosensitive robots to light source position

(b) Distances photosensitive robots to light source position

Figure 6.33: Scalability assessment in the light follower experiment. In both subfigures and for each swarm size, the darker curves represent the median evolution and the contours of the shadows are the first and third quantiles.

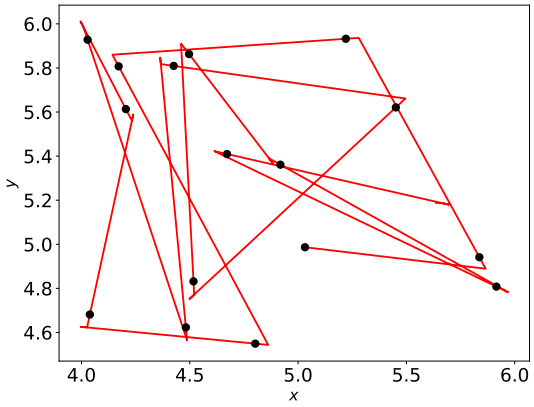
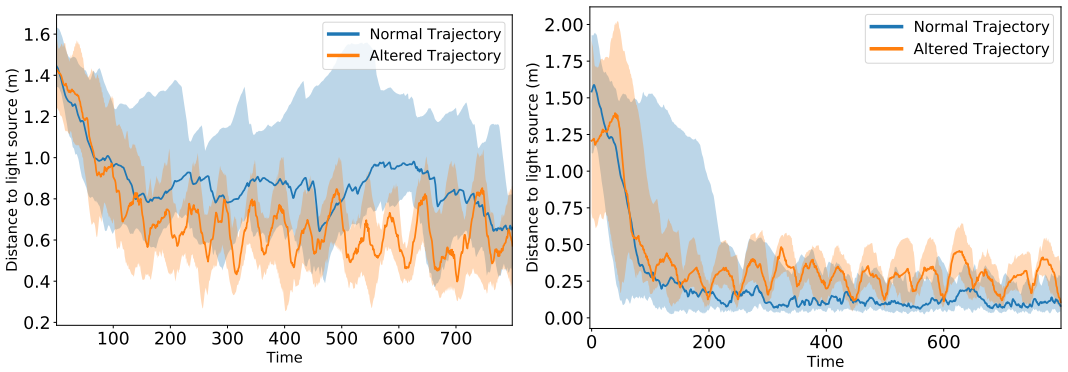


Figure 6.34: Example of altered trajectory of the light source used to assess the robustness in the light follower experiment. Red lines trace the light trajectory and black points show the sampled $\mathbf{x}_{tar}$ positions.

where α is a step size. More intuitively, the light source will describe linear trajectories between sampled $\mathbf{x}_{tar}$ points. When it reaches $\mathbf{x}_{tar}$ position, the light source oscillates around the target coordinates until a new position is sampled. Fig. 6.34 shows an example of the new trajectory of the light source, highlighting in black the sampled target positions. Once the trajectory is described, Figure 6.35 compares the performance in terms of the mean distance of the robots to the light for the novel and former trajectories. Curiously, the results show that the performance is slightly increased in the case of the robots with light sensor disabled and decreased when showing the distances of the photosensitive agents. However, in the light of the figures, it can be stated that the swarm behavior is generalizes properly with new light movement patters. It is also worth mentioning that the use of just few robots capable of sensing light is itself an evidence of robustness. Specifically, provided that there is at least one active photosensitive robot in the swarm, the group of robots can acceptably solve the task when a robot failure happens.

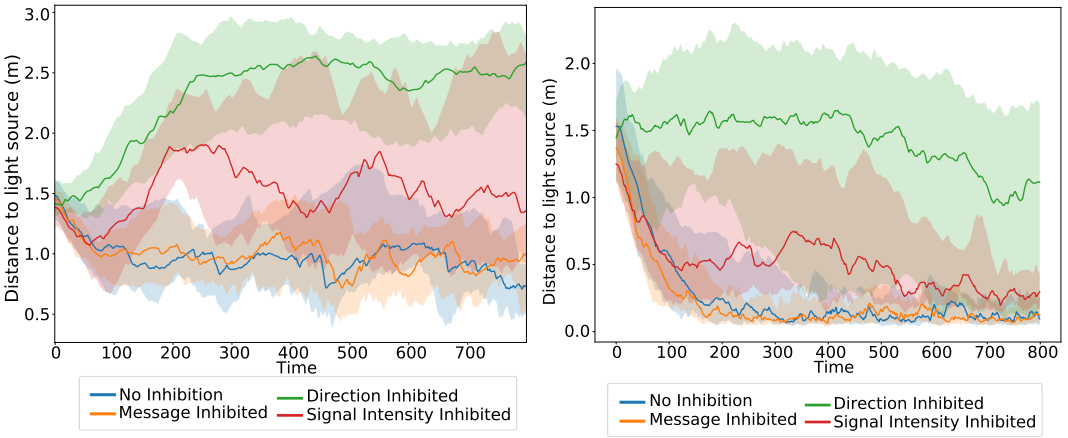
Communication: The communication mechanisms underlying the swarm behavior are



(a) Distances of non-photosensitive robots to light source position

(b) Distances of photosensitive robots to light source position

Figure 6.35: Robustness assessment. It compares the distributions of the distances to the light for the orbit trajectory used during evolution and the new altered trajectory (see Eq. 6.4).



(a) Distances of non-photosensitive robots to light source position

(b) Distances of photosensitive robots to light source position

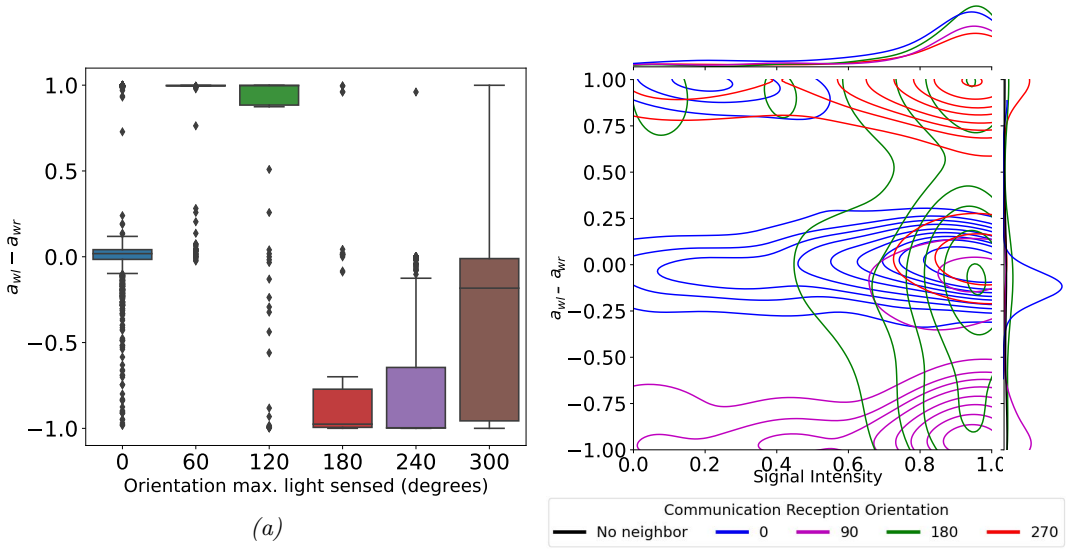
Figure 6.36: Comparison of distance $\mathrm{d}_{\mathcal{T}}$ *to the light position when different communication variables are inhibited and in normal conditions (no inhibition). In both subfigures and for each swarm size, the darker curves represent the median evolution and the contours of the shadows are the first and third quantiles.*

exposed hereafter. Firstly, the performance is tested when inhibiting or canceling each communication variable at CTRNN level. Fig. 6.36 shows how the deletion of the inputs affects on the mean distance to the light. Fig. 6.36a includes the robots that cannot sense the light while Fig. 6.36b depicts the photosensitive agent mean distances. The curves suggest that the message content is not relevant for solving the problem. In the light of the green and red curves, the orientation from where the message was received and the signal strength sensed are important communication variables used by the CTRNN in some manner. It should be mentioned that the state of the communication is not shown in the figure because it is set to send mode in the totality of the samples.

In order to analyze the relation between the communication information and the wheel actions, the difference between the right wheel action and the left wheel action, defined as $a_{wl} - a_{wr}$ is considered. Thereafter, taking into account the fact that, in this experiment, $a_{wl}, a_{wr} \in [0, 1]$ the following scenarios describe the possible robot movement patterns:

- If $a_{wl} - a_{wr} \approx 0$ then the robot approximately moves forward.
- If $a_{wl} - a_{wr} < 0$ then the robot turns to the left.
- If $a_{wl} - a_{wr} > 0$ then the robot turns to the right.
- In the two latter cases, the rotation will be more abrupt for larger values of $|a_{wl} - a_{wr}|$.

Fig. 6.37a shows boxplots representing the distribution of the wheel difference $a_{wl} - a_{wr}$ conditioned to the orientation from where the maximum light intensity was sensed. Only the data recorded from photosensible agents is used. Reasonably, when the agent measures maximum light intensity from its heading direction, at 0°, it goes forward. In the other situations, the wheel actions generally conduct the robot rotation in the correct direction, albeit there is high variance in the distributions. Moreover, Fig. 6.37b illustrates the joint



(a)

(b)

Figure 6.37: (a) Boxplots representing the distribution of $a_{wl} - a_{wr}$ for different orientations where maximum light intensity was measured. In the boxplots, the black line within the box is the median and each box encloses samples in between the first and third quantile, also known as the interquartile range. The whiskers extend to the farthest data points that are within 1.5 times the interquartile range. Outliers are shown as outer dots. (b) Kernel density estimate of the bivariate distribution of $a_{wl} - a_{wr}$ and the signal intensity of the received message. Each color represents the contour curves of the distributions for different message reception orientations.

distribution of the signal intensity of the received message and the wheel action difference conditioned to the orientation from where the message was received. The distributions, represented as contour plots, are approximated as the kernel density estimates. The curves located at the upper and right sides depict the estimations of the marginal distributions. The figure clearly shows that when the message sender is in front of the robot (blue contours), the agent moves straight forward in order to reach the sender's position. Alternatively, if the emitter is on its right side (in red) it turns right and when the sender is on its left side (in magenta) it turns left.

Chapter 7

Conclusions and Future Lines

7.1 Conclusions

This Master Thesis is framed within the intersection of the soft computing fields of swarm intelligence, evolutionary computation and artificial neural networks. Specifically, a set of robotics tasks were proposed and solved using a swarm of homogeneous simulated robots, controlled by Continuos-Time Recurrent Neural Networks (CTRNN). The neural controllers are optimized using the following evolutionary computation algorithms: genetic algorithm (GA) and Separable Natural Evolution Strategies (SNES). The experiments faced were the leader selection, swarm borderline identification, orientation consensus and light following tasks. The proposed experiments required, at some extent, the cooperative interaction and communication among robots, that are incapable of solving the problem individually. Therefore, we proposed a minimal communication system and, as one of the main objectives of the Master Thesis, analysed the emerged communication mechanics that result from evolution. In addition to the quantized real vector message, the communication also carries the context of the message (signal strength, orientation, and so on). Moreover, a communication state was designed so that an agent can decide on whether it sends its own message or relays the message that was received. The communication is minimal for several reasons: the communication is local in the sense an agent can only interact with other robots within a small neighboring range. Additionally, an agent can only communicate with one robot per time step and only the context information of the corresponding message can be acquired. Besides, the context information related to the orientation from where the message was sensed is highly discretized to four possible angles, utterly complexifying the interaction. All the required functionalities, robotics simulations, algorithms and neural models were also implemented in a software simulator as part of this Master Thesis.

The evolved controllers successfully solved each of the designed tasks with both GA and SNES. The only exception is the GA algorithm used in the leader selection experiment, which was not able to reach a good enough solution. SNES solved all the tasks with outstanding performance, and outperformed GA, in terms of mean fitness score, in all the experiments except in the borderline identification problem. In each experiment, apart from the behavior of the evolved swarm and its performance, we assessed the scalability and robustness capabilities of the solutions. These properties are utterly desirable in swarm robotics systems. The scalability was tested by varying the number of robots and observing the consequent behavioral and performance alterations. For the robustness, we designed a task specific perturbation or failure (e.g. robot fault or abrupt swarm topology change) and observed how the system responses and adapts to such undesired perturbations. In all the tasks, both scalability and robustness properties were, in general, correctly acquired, leading to a remarkably complete swarm robotics system that uses minimal communication means.

Furthermore, the communication that emerged from evolution was studied both using a single simulation trial and harnessing statistical tools with a large number of simulations, for the generation of reliable results. From this study, different types of communication emerged in each experiment. In the leader election task, the robots absolutely ignored the message context and purely rely on the communication state and the message, albeit the message is uniquely used as a binary flag to communicate that the robot is claiming the leadership. Subsequently, the observed communication in the case of the borderline identification is purely situated as only the context carries information. Within the orientation consensus problem the resulting interaction seems to rely both on the message and its context. It was observed that the context is employed by the CTRNN to match the heading orientation with its vicinity, while the message is harnessed for reducing the rotation speed once orientation consensus is achieved. Finally, the light follower task was accomplished with purely situated communication in which only the context of the message orientation and signal strength carry information.

7.2 Future lines

Several future lines of research have appeared during the development of this Master Thesis. Therefore, this subsection is dedicated to the enumeration and brief description of these improvements. We envision three bifurcations that lead to the progress of this work from different perspectives:

- **Cognitive and communication improvements**: encompasses the future lines devoted to explore the SR interaction capabilities, with minimalistic communication systems, and the limits of swarm robot's behaviors. Firstly, as it was already mentioned in Chapter 6, a deeper analysis of the precise communication mechanics that emerged in the experiments of this Master Thesis has been left as future work, paying special attention to the orientation consensus experiment. Besides, we have assessed the suitability of using almost the same neural controller in all the experiments, but with different and independent evolution trajectories. However, a remarkably interesting extension is the experiment of evolving the same recurrent neural network for all the proposed tasks, sequentially. This assessment would explore the limits of evolution and generalization capabilities of swarm neural controllers. Additionally, another important improvement is the addition of new SR tasks that require more abstract and complex communication mechanics and semantics. Finally, a research line to be faced in the future is related to the deepen into attractor theory and tools in recurrent neural networks. Although we already used the described theory of attractors in Chapter 2 for the justification of the two different time scales (environment and neuronal), there is still plenty of future research to be constructed on top of the presented theoretical concepts. We distinguish two different work trajectories under the frame of attractors. Firstly, the analysis of the emerged behaviors and communication at the neuronal level by observing the attractors and the attractor transitions within the state space of the network. Alternatively, a more appealing and challenging research line involves the design and implementation of learning or evolution algorithms that directly optimize the RNN at the attractor level.
- **Algorithmic improvements**: this bifurcation considers all the system enhancements related to the employment of new optimization algorithms and artificial neural networks. A particularly attractive evolution algorithm to be explored is the NeuroEvolution of Augmenting Topologies (NEAT) algorithm (see [79]), which optimizes not only the weights of the synapses but also the neural architecture itself. Note that the use of NEAT can be combined with the previously mentioned future line of evolving

the same neural controller for task generalization. In this situation, it could be studied how the algorithm extends or reduces the neural architecture in order to adapt its policy to new tasks. Another important algorithm addition that is worth exploring is the use of novelty search (see [47] and [81]) for the emergence of richer and more complex swarm behaviors. Novelty search rewards agents for exploring new areas of the behavioral space instead of using the fitness function as usual. Apart from the research lines related to other evolutionary computation algorithms, the exploration of other recurrent neural networks is also important. Specifically, building and evolving neural controllers of spiking neural networks is a remarkable future line. Firslty, spiking neural networks have been scarcely explored in swarm robotics and reinforcement learning experiments. Furthermore, they are biologically realistic, highly powerful in tasks where time is present and can be outstandingly efficient and fast in dedicated neuromorphic hardware [101]. Spiking neural networks lead to the last algorithmic future line to be considered, which is the combination of evolution with lifetime development of the individuals, from a Baldwinian prospect. The lifetime learning in spiking neural networks can be accomplished by means of local Hebbian based rules, such as Spike-Timing-Dependent Plasticity (STDP) or three factor rules.

- **Software improvements**: the upgrades of the software simulator to be considered in future versions are exposed in detail in Section 4.5. The most relevant ones are the incorporation of 3D graphics and 3D physics, the exploration of new optimization algorithms and the addition of other parallelization mechanics.

Bibliography

[1] Gerardo Beni. From swarm intelligence to swarm robotics. SAB'04, page 1–9, Berlin, Heidelberg, 2004. Springer-Verlag.

[2] Eric Bonabeau, Marco Dorigo, and Guy Theraulaz. *From Natural to Artificial Swarm Intelligence*. Oxford University Press, Inc., USA, 1999.

[3] Erol Sahin. Swarm robotics: From sources of inspiration to domains of application. In *Swarm Robotics*, pages 10–20, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[4] Levent Bayındır. A review of swarm robotics tasks. *Neurocomputing*, 172:292 – 321, 2016.

[5] Manuele Brambilla, Eliseo Ferrante, Mauro Birattari, and Marco Dorigo. Swarm robotics: A review from the swarm engineering perspective. *Swarm Intelligence*, 7:1–41, 03 2013.

[6] Ying Tan and Zhong yang Zheng. Research advance in swarm robotics. *Defence Technology*, 9(1):18 – 39, 2013.

[7] Vito Trianni, Roderich Gross, Thomas H. Labella, Erol Sahin, and Marco Dorigo. Evolving aggregation behaviors in a swarm of robots. In *Advances in Artificial Life*, pages 865–874, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[8] O. Soysal and E. Sahin. Probabilistic aggregation strategies in swarm robotic systems. In *Proceedings 2005 IEEE Swarm Intelligence Symposium, 2005. SIS 2005.*, pages 325–332, 2005.

[9] Melvin Gauci, Jianing Chen, Tony J. Dodd, and Roderich Gross. Evolving aggregation behaviors in multi-robot systems with binary sensors. In *Distributed Autonomous Robotic Systems*, pages 355–367, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[10] Ziya Firat, Eliseo Ferrante, Yannick Gillet, and Elio Tuci. On self-organised aggregation dynamics in swarms of robots with informed robots. *Neural Computing and Applications*, 32(17):13825–13841, 2020.

[11] Craig W. Reynolds. Flocks, herds and schools: A distributed behavioral model. *SIGGRAPH Comput. Graph.*, 21(4):25–34, 1987.

[12] Rita Parada Ramos, Sancho Moura Oliveira, Susana Margarida Vieira, and Anders Lyhne Christensen. Evolving flocking in embodied agents based on local and global application of reynolds' rules. *PLOS ONE*, 14(10):1–16, 10 2019.

[13] Ali Turgut, Hande Çelikkanat, Fatih Gökçe, and Erol Sahin. Self-organized flocking in mobile robot swarms. *Swarm Intelligence*, 2:97–120, 12 2008.

[14] Alfio Borzì and Suttida Wongkaew. Modeling and control through leadership of a refined flocking system. *Mathematical Models and Methods in Applied Sciences*, 25:255–282, 02 2015.

[15] Eliseo Ferrante, Ali Emre Turgut, Nithin Mathews, Mauro Birattari, and Marco Dorigo. Flocking in stationary and non-stationary environments: A novel communication strategy for heading alignment. In *Parallel Problem Solving from Nature, PPSN XI*, pages 331–340, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[16] Joshua P. Hecker and Melanie E. Moses. Beyond pheromones: evolving error-tolerant, flexible, and scalable ant-inspired robot swarms. *Swarm Intelligence*, 9:43–70, 2015.

[17] J. Ericksen, M. Moses, and S. Forrest. Automatically evolving a general controller for robot swarms. In *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–8, 2017.

[18] Yong Song, Xing Fang, Bing Liu, Caihong Li, Yibin Li, and Simon X. Yang. A novel foraging algorithm for swarm robotics based on virtual pheromones and neural network. *Applied Soft Computing*, 90:106156, 2020.

[19] Alexandre Campo and Marco Dorigo. Efficient multi-foraging in swarm robotics. In *Advances in Artificial Life*, pages 696–705, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[20] Alexandre Campo, Álvaro Gutiérrez, Shervin Nouyan, Carlo Pinciroli, Valentin Longchamp, Simon Garnier, and Marco Dorigo. Artificial pheromone for path selection by a foraging swarm of robots. *Biological Cybernetics*, 103:339–352, 2010.

[21] Thomas H. Labella, Marco Dorigo, and Jean-Louis Deneubourg. Division of labor in a group of robots inspired by ants' foraging behavior. *ACM Trans. Auton. Adapt. Syst.*, 1(1):4–25, 2006.

[22] James Wilson, Jon Timmis, and Andy Tyrrell. A hormone arbitration system for energy efficient foraging in robot swarms. In *Towards Autonomous Robotic Systems*, pages 305–316, Cham, 2018. Springer International Publishing.

[23] Javier de Lope, Darío Maravall, and Yadira Quiñonez. Self-organizing techniques to improve the decentralized multi-task distribution in multi-robot systems. *Neurocomputing*, 163:47–55, 2015.

[24] Giovanni Pini, Arne Brutschy, Marco Frison, Andrea Roli, Marco Dorigo, and Mauro Birattari. Task partitioning in swarms of robots: an adaptive method for strategy selection. *Swarm Intelligence*, 5:283–304, 2011.

[25] B. Pang, C. Zhang, Y. Song, and H. Wang. Self-organized task allocation in swarm robotics foraging based on dynamical response threshold approach. In *2017 18th International Conference on Advanced Robotics (ICAR)*, pages 256–261, 2017.

[26] Lorenzo Garattoni and Mauro Birattari. Autonomous task sequencing in a robot swarm. *Science Robotics*, 3(20), 2018.

[27] Arne Brutschy, Giovanni Pini, Carlo Pinciroli, Mauro Birattari, and Marco Dorigo. Self-organized task allocation to sequentially interdependent tasks in swarm robotics. *Autonomous Agents and Multi-Agent Systems*, 28:101–125, 2014.

[28] Manuel Castillo-Cagigal, Arne Brutschy, Alvaro Gutiérrez, and Mauro Birattari. Temporal task allocation in periodic environments. In *Swarm Intelligence*, pages 182–193, Cham, 2014. Springer International Publishing.

[29] Michael Rubenstein, Alejandro Cornejo, and Radhika Nagpal. Programmable self-assembly in a thousand-robot swarm. *Science*, 345(6198):795–799, 2014.

[30] Mehmet Serdar Güzel and Hakan Kayakökü. A collective behaviour framework for multi-agent systems. In *Mechatronics and Robotics Engineering for Advanced and Intelligent Manufacturing*, pages 61–71, Cham, 2017. Springer International Publishing.

[31] Valery Karpov and Irina Karpova. Leader election algorithms for static swarms. *Biologically Inspired Cognitive Architectures*, 12:54 – 64, 2015.

[32] Joshua Cherian Varughese, Hannes Hornischer, Payam Zahadat, Ronald Thenius, Franz Wotawa, and Thomas Schmickl. A swarm design paradigm unifying swarm behaviors using minimalistic communication. *Bioinspiration & Biomimetics*, 15(3):036005, 2020.

[33] Michael Rubenstein, Adrian Cabrera, Justin Werfel, Golnaz Habibi, James McLurkin, and Radhika Nagpal. Collective transport of complex objects by simple robots: Theory and experiments. In *Proceedings of the 2013 International Conference on Autonomous Agents and Multi-Agent Systems*, AAMAS '13, page 47–54, Richland, SC, 2013. International Foundation for Autonomous Agents and Multiagent Systems.

[34] Serge Kernbach, Dagmar Häbe, Olga Kernbach, Ronald Thenius, Gerald Radspieler, Toshifumi Kimura, and Thomas Schmickl. Adaptive collective decision making in limited robot swarms without communication. *The International Journal of Robotics Research*, 32:35–55, 01 2013.

[35] William M. Spears, Diana F. Spears, Jerry C. Hamann, and Rodney Heil. Distributed, physics-based control of swarms of vehicles. *Autonomous Robots*, 17:137–162, 2004.

[36] Attilio Priolo. *Swarm aggregation algorithms for multi-robot systems.* PhD thesis, University of Roma Tre, 2013.

[37] Stefano Nolfi and Dario Floreano. *Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines.* The MIT Press, Cambridge, MA., 2000.

[38] Dario Floreano, Peter Dürr, and Claudio Mattiussi. Neuroevolution: from architectures to learning. *Evolutionary Intelligence*, 1(1):47–62, 2008.

[39] G. Baldassarre, V. Trianni, M. Bonani, F. Mondada, M. Dorigo, and S. Nolfi. Self-organized coordinated motion in groups of physically connected robots. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 37(1):224–239, 2007.

[40] Gianluca Baldassarre, Stefano Nolfi, and Domenico Parisi. Evolving mobile robots able to display collective behaviors. *Artificial life*, 9:255–67, 02 2003.

[41] Randall D. Beer and John C. Gallagher. Evolving dynamical neural networks for adaptive behavior. *Adapt Behav*, 1(1):91–122, 1992.

[42] Elio Tuci, Boris Mitavskiy, Stefano Benedettini, and Gianpiero Francesca. On the evolution of self-organised role-allocation and role-switching behaviour in swarm robotics: a case study. pages 379–386, 12 2012.

[43] Alvaro Gutiérrez, Elio Tuci, and Alexandre Campo. Evolution of neuro-controllers for robots' alignment using local communication. *International Journal of Advanced Robotic Systems*, 6, 03 2009.

[44] Muhanad Alkilabi, Aparajit Narayan, and Elio Tuci. Cooperative object transport with a swarm of e-puck robots: robustness and scalability of evolved collective strategies. *Swarm Intelligence*, 11:185–209, 2017.

[45] D.E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley Longman Publishing Co., Inc., USA, 1989.

[46] Eric Medvet, Stefano Seriani, Alberto Bartoli, and Paolo Gallina. Design of powered floor systems for mobile robots with differential evolution. In *Applications of Evolutionary Computation*, pages 19–32, Cham, 2019. Springer International Publishing.

[47] Jorge Gomes, Paulo Urbano, and Anders Lyhne Christensen. Introducing novelty search in evolutionary swarm robotics. In *Swarm Intelligence*, pages 85–96, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[48] Miguel Duarte, Jorge Gomes, Vasco Costa, Sancho Moura Oliveira, and Anders Lyhne Christensen. Hybrid control for a real swarm robotics system in an intruder detection task. In *Applications of Evolutionary Computation*, pages 213–230, Cham, 2016. Springer International Publishing.

[49] Daan Wierstra, Tom Schaul, Tobias Glasmachers, Yi Sun, Jan Peters, and Jürgen Schmidhuber. Natural evolution strategies. *J. Mach. Learn. Res.*, 15(1):949–980, 2014.

[50] Tom Schaul, Tobias Glasmachers, and Jürgen Schmidhuber. High dimensions and heavy tails for natural evolution strategies. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, GECCO '11, page 845–852. Association for Computing Machinery, 2011.

[51] Y. Uny Cao, Alex S. Fukunaga, and Kahng Andrew. Cooperative mobile robotics: antecedents and directions. *Autonomous Robots*, 4:7–27, 1997.

[52] Farshad Arvin, Khairulmizam Samsudin, and Abdul Ramli. Development of ir-based short-range communication techniques for swarm robot applications. *Advances in Electrical and Computer Engineering*, 10(4):61–68, 2010.

[53] Serge Kornienko, Olga Kornienko, and P. Levi. Ir-based communication and perception in microrobotic swarms. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'05)*, 2005.

[54] Gutiérrez Alvaro, Alexandre Campo, Marco Dorigo, Amor Daniel, Luis Magdalena, and Monasterio-Huelin Félix. An open localization and local communication embodied sensor. *Sensors*, 8:7545–7563, 2008.

[55] Onofrio Gigliotta, Marco Mirolli, and Stefano Nolfi. Communication based dynamic role allocation in a group of homogeneous robots. *Natural Computing*, 13:391–402, 2014.

[56] S. Kornienko, O. Kornienko, and P. Levi. Collective ai: Context awareness via communication. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, IJCAI'05, page 1464–1470, San Francisco, CA, USA, 2005. Morgan Kaufmann Publishers Inc.

[57] Baoding Zhang and Shulan Gao. The study of zigbee technology's application in swarm robotics system. In *2011 2nd International Conference on Artificial Intelligence, Management Science and Electronic Commerce (AIMSEC)*, pages 1763–1766, 2011.

[58] Ulf Witkowski and Reza Zandian. Novel method of communication in swarm robotics based on the nfc technology. pages 377–389, 06 2014.

[59] Türker Türkoral, Özgür Tamer, Suat Yetiş, and Levent Çetin. Indoor localization for swarm robotics with communication metrics without initial position information. In *Mechatronics and Robotics Engineering for Advanced and Intelligent Manufacturing*, pages 207–215, Cham, 2017. Springer International Publishing.

[60] Elio Tuci and Christos Ampatzis. Evolution of acoustic communication between two cooperating robots. volume 4648, pages 395–404, 09 2007.

[61] Kasper Støy. Using situated communication in distributed autonomous mobile robotics. In *Proceedings of the Seventh Scandinavian Conference on Artificial Intelligence*, SCAI '01, page 44–52, NLD, 2001. IOS Press.

[62] Joachim de Greeff and Stefano Nolfi. *Evolution of Implicit and Explicit Communication in Mobile Robots*, pages 179–214. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[63] Tiago Rodrigues, Miguel Duarte, Sancho Oliveira, and Anders Lyhne Christensen. Beyond onboard sensors in robotic swarms. In *Proceedings of the International Conference on Agents and Artificial Intelligence - Volume 2*, ICAART 2015, page 111–118, Setubal, PRT, 2015. SCITEPRESS - Science and Technology Publications, Lda.

[64] Ah Chung Tsoi and Andrew Back. Discrete time recurrent neural network architectures: A unifying review. *Neurocomputing*, 15(3):183–223, 1997.

[65] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.

[66] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation, 2014.

[67] Bard Ermentrout and David Terman. *The Mathematical Foundations of Neuroscience*, volume 35. 07 2010.

[68] Wulfram Gerstner and Werner M. Kistler. *Spiking Neuron Models: Single Neurons, Populations, Plasticity*. Cambridge University Press, 2002.

[69] Gabriel Kreiman. Neural coding: computational and biophysical perspectives. *Physics of Life Reviews*, 1(2):71–102, 2004.

[70] Simon S. Haykin. *Neural Networks and Learning Machines*. Pearson Education, third edition, 2009.

[71] R. Chaudhuri and I Fiete. Computational principles of memory. *Nature Neuroscience*, 19(3):394–403, 2016.

[72] Razvan Pascanu and Herbert Jaeger. A neurodynamical model for working memory. *Neural Networks*, 24(2):199 – 207, 2011.

[73] Mikhail Rabinovich and Pablo Varona. Robust transient dynamics and brain functions. *Frontiers in Computational Neuroscience*, 5:24, 2011.

[74] Mikhail I. Rabinovich and Pablo Varona. Discrete sequential information coding: Heteroclinic cognitive dynamics. *Frontiers in Computational Neuroscience*, 12:73, 2018.

[75] J J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, 79(8):2554–2558, 1982.

[76] C. Eliasmith. A unified approach to building and controlling spiking attractor networks. *Neural Computation*, 17(6):1276–1314, 2005.

[77] Faustino Gomez, Jürgen Schmidhuber, and Risto Miikkulainen. Efficient non-linear control through neuroevolution. In *Machine Learning: ECML 2006*, pages 654–662, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[78] Faustino J. Gomez. *Robust Non-Linear Control through Neuroevolution.* PhD thesis, Department of Computer Sciences, The University of Texas at Austin, 2003.

[79] Kenneth O. Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evol. Comput.*, 10(2):99–127, 2002.

[80] Kenneth O. Stanley, David B. D'Ambrosio, and Jason Gauci. A hypercube-based encoding for evolving large-scale neural networks. *Artif. Life*, 15(2):185–212, 2009.

[81] Edoardo Conti, Vashisht Madhavan, Felipe Petroski Such, Joel Lehman, Kenneth Stanley, and Jeff Clune. Improving exploration in evolution strategies for deep reinforcement learning via a population of novelty-seeking agents. In *Advances in Neural Information Processing Systems*, volume 31, pages 5027–5038. Curran Associates, Inc., 2018.

[82] Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. Evolution strategies as a scalable alternative to reinforcement learning, 2017.

[83] B. Miller and D. Goldberg. Genetic algorithms, tournament selection, and the effects of noise. *Complex Syst.*, 9, 1995.

[84] Larry J. Eshelman and J. David Schaffer. Real-coded genetic algorithms and interval-schemata. In *Foundations of Genetic Algorithms*, volume 2, pages 187 – 202. Elsevier, 1993.

[85] Francisco Herrera, Manuel Lozano, E. Pérez, A. M. Sánchez, and P. Villar. Multiple crossover per couple with selection of the two best offspring: An experimental study with the blx-alpha crossover operator for real-coded genetic algorithms. In *Proceedings of the 8th Ibero-American Conference on AI: Advances in Artificial Intelligence*, page 392–401, 2002.

[86] Shun-Ichi Amari. Natural gradient works efficiently in learning. *Neural Comput.*, 10(2):251–276, 1998.

[87] Tobias Glasmachers, Tom Schaul, Sun Yi, Daan Wierstra, and Jürgen Schmidhuber. Exponential natural evolution strategies. In *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation*, GECCO '10, page 393–400. Association for Computing Machinery, 2010.

[88] Michel Marie Deza and Elena Deza. *Encyclopedia of Distances*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[89] N. Koenig and A. Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, volume 3, pages 2149–2154, 2004.

[90] Olivier Michel. Cyberbotics ltd. webots™: Professional mobile robot simulation. *International Journal of Advanced Robotic Systems*, 1(1):5, 2004.

[91] Carlo Pinciroli, Vito Trianni, Rehan O'Grady, Giovanni Pini, Arne Brutschy, Manuele Brambilla, Nithin Mathews, Eliseo Ferrante, Gianni Di Caro, Frederick Ducatelle, Mauro Birattari, Luca Maria Gambardella, and Marco Dorigo. Argos: a modular, parallel, multi-engine simulator for multi-robot systems. *Swarm Intelligence*, 6(4):271–295, 2012.

[92] Trevor Bekolay, James Bergstra, Eric Hunsberger, Travis DeWolf, Terrence Stewart, Daniel Rasmussen, Xuan Choo, Aaron Voelker, and Chris Eliasmith. Nengo: a python tool for building large-scale functional brain models. *Frontiers in Neuroinformatics*, 7:48, 2014.

[93] Dan Goodman and Romain Brette. The brian simulator. *Frontiers in Neuroscience*, 3:26, 2009.

[94] Hananel Hazan, Daniel J. Saunders, Hassaan Khan, Devdhar Patel, Darpan T. Sanghavi, Hava T. Siegelmann, and Robert Kozma. Bindsnet: A machine learning-oriented spiking neural networks library in python. *Frontiers in Neuroinformatics*, 12:89, 2018.

[95] Félix-Antoine Fortin, François-Michel De Rainville, Marc-André Gardner Gardner, Marc Parizeau, and Christian Gagné. Deap: Evolutionary algorithms made easy. *J. Mach. Learn. Res.*, 13(1):2171–2175, July 2012.

[96] David E. Moriarty. *Symbiotic Evolution Of Neural Networks In Sequential Decision Tasks*. PhD thesis, Department of Computer Sciences, The University of Texas at Austin, 1997.

[97] Jackie Shen. Cucker–smale flocking under hierarchical leadership. *SIAM Journal of Applied Mathematics*, 68:694–719, 01 2007.

[98] Matt Quinn, Lincoln Smith, Giles Mayley, and Phil Husbands. Evolving controllers for a homogeneous system of physical robots: Structured cooperation with minimal sensors. *Philosophical transactions. Series A, Mathematical, physical, and engineering sciences*, 361:2321–43, 11 2003.

[99] Ovunc Tuzel, Gilberto Marcon dos Santos, Chloë Fleming, and Julie A. Adams. Learning based leadership in swarm navigation. In *Swarm Intelligence*, pages 385–394, Cham, 2018. Springer International Publishing.

[100] Herbert Edelsbrunner. Alpha shapes—a survey. *Tessellations in the Sciences*, 01 2010.

[101] A. R. Young, M. E. Dean, J. S. Plank, and G. S. Rose. A review of spiking neuromorphic hardware communication systems. *IEEE Access*, 7:135606–135620, 2019.

[102] Robin Murphy, Satoshi Tadokoro, Daniele Nardi, Adam Jacoff, Paolo Fiorini, Howie Choset, and Aydan Erkmen. *Search and Rescue Robotics*, pages 1151–1173. 01 2008.

[103] M. Bakhshipour, M. Jabbari Ghadi, and F. Namdari. Swarm robotics search and rescue: A novel artificial intelligence-inspired optimization approach. *Applied Soft Computing*, 57:708 – 726, 2017.

[104] Mauro S. Innocente and Paolo Grasso. Self-organising swarms of firefighting drones: Harnessing the power of collective intelligence in decentralised multi-robot systems. *Journal of Computational Science*, 34:80 – 101, 2019.

[105] Gustavo Pessin, DanielO Sales, Mauricio Dias, Rafael Klaser, DenisF Wolf, Jó Ueyama, Fernando Osorio, and PatríciaA Vargas. Swarm intelligence and the quest to solve a garbage and recycling collection problem. *Soft Computing*, 17:1–15, 2013.

Appendix A

Example of a configuration file

```
    "checkpoint_file" : "chk_border_GA",
    "topology" : {
        "dt" : 0.1,
        "time_scale" : 30,
        "stimuli": {
            "InputA" : {"n" : 2, "sensor" : "wireless_receiver:msg"},
            "InputB" : {"n" : 1, "sensor" : "wireless_receiver:state"},
            "InputC" : {"n" : 2, "sensor" : "wireless_receiver:receiving_direction"}
        },
        "encoding" : {
            "InputA" :{"scheme" : "IdentityEncoding"},
            "InputB" :{"scheme" : "IdentityEncoding"},
            "InputC" :{"scheme" : "IdentityEncoding"}
        },
        "neurons" : {"model": "ctrnn"},
        "ensembles": {
            "H1" : {"n" : 10, "activation":"sigmoid"},
            "H2" : {"n" : 10, "activation":"sigmoid"},
            "OUT_COMM" : {"n" : 2,  "activation":"sigmoid"},
            "OUT_COMM_STATE" : {"n" : 1, "activation":"sigmoid"},
            "OUT_LED" : {"n" : 1, "activation":"sigmoid"}
        },
        "outputs" : {
            "outA" : {"ensemble" : ["OUT_COMM"], "actuator" : "wireless_transmitter",
                ↪ "enc": "real"},
            "outC" : {"ensemble" : ["OUT_COMM_STATE"], "actuator" :
                ↪ "wireless_transmitter:state", "enc": "cat"},
            "outB" : {"ensemble" : ["OUT_LED"], "actuator" : "led_actuator", "enc": "cat"}
        },
        "synapses" :  {
            "IA-H1" : {"pre":"InputA","post":"H1", "trainable":true, "p":1},
            "IB-H1" : {"pre":"InputB","post":"H1", "trainable":true, "p":1},
            "IC-H1" : {"pre":"InputC","post":"H1", "trainable":true, "p":1},
            "IA-H2" : {"pre":"InputA","post":"H2", "trainable":true, "p":1},
            "IB-H2" : {"pre":"InputB","post":"H2", "trainable":true, "p":1},
            "IC-H2" : {"pre":"InputC","post":"H2", "trainable":true, "p":1},
            "H1-H1" : {"pre":"H1","post":"H1", "trainable":true, "p":0.7},
            "H2-H2" : {"pre":"H2","post":"H2", "trainable":true, "p":0.7},
            "H1-H2" : {"pre":"H1","post":"H2", "trainable":true, "p":0.7},
            "H1-led" : {"pre":"H1","post":"OUT_LED", "trainable":true, "p":1},
            "H2-comm" : {"pre":"H2","post":"OUT_COMM", "trainable":true, "p":1},
            "H2-st" : {"pre":"H2","post":"OUT_COMM_STATE", "trainable":true, "p":1},
            "comm-H1" : {"pre":"OUT_COMM","post":"H1", "trainable":true, "p":0.85},
            "led-H1" : {"pre":"OUT_LED","post":"H1", "trainable":true, "p":0.85},
            "st-H1" : {"pre":"OUT_COMM_STATE","post":"H1", "trainable":true, "p":0.85},
            "comm-comm" : {"pre":"OUT_COMM","post":"OUT_COMM", "trainable":true, "p":1}
        },
        "decoding" : {
            "outA" : {"scheme" : "IdentityDecoding", "params" : {"is_cat" : false}},
            "outC" : {"scheme" : "ThresholdDecoding", "params" : {"is_cat" : true}},
            "outB" : {"scheme" : "ThresholdDecoding", "params" : {"is_cat" : true}}
        }
    },
    "algorithm" : {
        "name" : "GA",
        "evolvable_object" : "robotA",
        "population_size" : 100,
        "generations" : 1000,
```

```
        "evaluation_steps" : 600,
        "num_evaluations" : 5,
        "fitness_function" : "identify_borderline",
        "populations" : {
            "p1" : {
                "objects" : ["synapses:weights:all", "neurons:bias:all",
                    ↪ "neurons:tau:all"],
                "max_vals" : [3,  2, 0.75],
                "min_vals" : [-3, -2, -1],
                "encoding" : "real",
                "selection_operator" : "tournament",
                "crossover_operator" : "blxalpha",
                "mutation_operator" : "gaussian",
                "mating_operator" : "random",
                "mutation_prob" : 0.05,
                "crossover_prob" : 0.9,
                "num_elite" : 3
            }
        }
    },
    "world":{
        "world_delay" : 1,
        "render_connections":true,
        "height" : 1000,
        "width" : 1000,
        "objects" : {
            "robotA" : {
                "type" : "robot",
                "num_instances" : 20,
                "controller" : "neural_controller",
                "sensors" : {"wireless_receiver" : {"n_sectors" : 4, "range" : 80,
                    ↪ "msg_length" : 2}},
                "actuators" : {"led_actuator" : {}, "wireless_transmitter" : {"quantize":
                    ↪ true, "range" : 80, "msg_length":2}},
                "initializers" : {
                    "positions" : {"name" : "random_graph", "params" : {"max_rad":100,
                        ↪ "initial_pos" : [500, 500]}},
                    "orientations" : {"name" : "random_uniform",  "params" : {"low" : 0,
                        ↪ "high" : 6.28, "size" : 1}}
                },
                "params" : {"trainable" : true}
            }
        }
    }
}
```

Appendix B

Ethical, economical, social and environmental aspects

B.1 Introduction

This Appendix describes the potential impacts that this Master Thesis can have in terms of ethical, economical, social and environmental aspects. Note that, owing to the fact that this Master Thesis is envisioned from a theoretical perspective, and all the experiments are simulated, it is difficult to highlight direct impacts of this work. Clearly, there is a direct impact on educational and research aspects due to the designed SR framework and the implemented software simulator. Furthermore, the designed communication system and the recurrent neural network evolution can be used in the future to build reliable and robust robotic systems that directly impact on the society.

B.2 Description of the relevant project related problems

B.2.1 Social impact

The use of the designed swarm robotics paradigm can have a straight social impact because it can improve existing swarm robotics tasks of seek and rescue (see e.g. [102] and [103]). In these tasks, swarms of robots generally have to coordinate in order to seek and rescue victims of a disaster. Note that the tasks addressed here, such as leader election and orientation consensus, are highly relevant for the efficient accomplishment of the seek and rescue problem. Additionally, other potential uses of our work devoted to cover social requirements are transportation, food supply or surveillance, among others.

B.2.2 Economical impact

The use of our designed SR system (communication, controller optimization, and so on) has a clear economical impact. Specifically, as stated throughout all this document, the system is designed to be simple yet robust and scalable, leading to potential cost reduction of mobile robotics systems (hardware and software). Moreover, multiple swarm robotics applications are related to task automation, highly reducing the operative costs.

B.2.3 Ethical impact

Although we are not aware of potential applications of our work in the ethical aspect, it is expected that several ideas and use cases arise in the future due to the constant growth of

swarm robotics field.

B.2.4 Environmental impact

There are also plenty of examples of use cases of our system that can pertain environmental impact. Firstly, applications devoted to fire prevention or contingency (see e.g. [104]) in forests are a notable example. Another example is garbage collection using swarm of robots (see e.g. [105]), which is equally environmentally impactful.

B.3 Conclusions

As it has been exposed, the use of the swarm robotics system designed and elaborated in this Master Thesis has multiple potential use cases that can remarkably impact on social, economical and environmental aspects. Moreover, there is also a clear contribution to the research of swarm robotics and multi-agent systems. Similarly, the designed and implemented robotics simulator can have great impact on the educational aspect.

Appendix C

Economic budget

Table. C.1 displays the economical budget associated to this Master Thesis.

		Cost of labor (Direct cost)		
		Hours	**Price/hour**	**Total**
		400	30€	**12.000 €**
Material costs (Direct cost)				
	Price	**Use in months**	**Amortization (years)**	**Total**
Compute server	4000	6	4	500 €
Total material costs				**500 €**
Total Costs				
Cost of labor				**12.000 €**
Material costs				**500 €**
General expenses (Indirect cost)	15% of direct cost			**1.875 €**
Industrial benefits	6% of direct and indirect costs			**862,5 €**
IVA (21%)				**3.199,875 €**
Total				**18.437,38 €**

Table C.1: Economical budget associated to the project.