UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS DE TELECOMUNICACIÓN

MÁSTER UNIVERSITARIO EN INGENIERÍA DE TELECOMUNICACIÓN

MASTER THESIS

Design and Development of Recurrent Neural Controllers for the Emergence of Communication in a Multi-Agent Cooperative System Using Evolutionary Computation.

Rafael Sendra Arranz

2021

UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS DE TELECOMUNICACIÓN

MÁSTER UNIVERSITARIO EN INGENIERÍA DE TELECOMUNICACIÓN

MASTER THESIS

Design and Development of Recurrent Neural Controllers for the Emergence of Communication in a Multi-Agent Cooperative System Using Evolutionary Computation.

Author
Rafael Sendra Arranz

Tutor
Álvaro Gutiérrez Martín

2021

Abstract

In multi-agent systems, multiple self-sufficient agents autonomously and intelligently cooperate or compete in a distributed way in order to solve complex tasks. Collective robotics is a type of cooperative multi-agent system in which simple robots locally interact in order to achieve a common goal. Plenty of studies have tackled individual collective robotics problems, such as flocking, foraging or aggregation. Nonetheless, a complex topic that has not been sufficiently explored yet is the problem of multitasking and task switching. Under the frame of collective robotics, task switching means that the robots have to sequentially cope with multiple, different and sometimes unrelated tasks as a group. Moreover, they have to be able to swap between the correct tasks at the precise timing. A particularly interesting perspective to this topic proposes that the multitasking and task switching behaviors of the robots must be obtained as a result of an artificial evolution process, starting from zero. One of the three principal goals of this Master Thesis is precisely the design, implementation and evaluation of a collective robotics system that is optimized using evolutionary computation in order to proficiently learn to solve and switch between two unrelated tasks. Specifically, the NeuroEvolution of Augmenting Topologies (NEAT) algorithm is employed as optimization procedure of the Continuous-Time Recurrent Neural Networks that control the robot's actions. We combine the evolution process of NEAT with lifetime learning using Generalized ABCD Hebbian learning rules. Aside from the task switching, another remarkably important subfield of collective robotics is the communication among robots. Combined with the multitasking and task switching between two tasks and using the previously mentioned algorithms, we also address this topic in this Master Thesis. It is posed in an experiment in which only one robot is aware of the task that has to be solved. Thereafter, this robot has to learn how to notify the correct information to the rest of the group. We propose an IR based minimal communication system that the robots can harness, albeit the communication semantics have to emerge from the evolutionary process. The third equally important pillar of this Master Thesis is the improvement and radical modification of an evolutionary robotics simulator previously developed by us. The most significant upgrades are the addition of 3D graphics, collision detection and realistic physics (using the python's pybullet library) and the implementation of the algorithms used in the collective robotics tasks. The structure of the Master Thesis consists of two experiments treating the collective robotics problems of task switching and communication emergence. After evolution, it is shown that robots are able to successfully address both experiments with outstanding results. In order to support the results, we use an statistically significant sample of 50 independent simulations.

Keywords— Collective Robotics, Evolutionary Computation, Continuous-Time Recurrent Neural Networks, Recurrent Neural Networks, Communication Emergence, Minimal Communication, Task Switching, MultiTasking, Neuroevolution of Augmenting Topologies, Hebbian Learning, Neural Controller, Neuroevolution, Evolutionary Robotics Simulator.

Resumen

En los sistemas multi-agente, múltiples agentes autosuficientes cooperan o compiten de forma autónoma, distribuida e inteligente con fin de resolver tareas complejas. La robótica colectiva es un tipo de sistema multi-agente cooperativo en el que robots simples interaccionan localmente para alcanzar un objetivo común. Una gran variedad de estudios han abordado tareas individuales de robótica colectiva, como por ejemplo flocking, foraging o agregación. No obstante, un tema complejo que no ha sido suficientemente explorado aún es el problema de multitarea y cambio de tarea. Bajo el marco de la robótica colectiva, cambio de tarea significa que los robots deben resolver secuencialmente múltiples tareas diferentes y, en algunos casos, no relacionadas. Además, el grupo de robots debe ser capaz de hacer el cambio entre tareas correctamente y en el momento preciso. Una perspectiva particularmente interesante de este problema plantea que la habilidad de realizar múltiples tareas y cambiar entre ellas debe de obtenerse como resultado de un proceso evolutivo artificial, comenzando desde cero. Uno de los tres pilares de esta Tesis de Máster es precisamente el diseño, implementación y evaluación de un sistema de robótica colectiva optimizado usando computación evolutiva con el fin de aprender a resolver y cambiar entre dos tareas no relacionadas. En concreto, el algoritmo NeuroEvolution of Augmenting Topologies (NEAT) es empleado como proceso de optimización de redes neuronales recurrentes en tiempo continuo, responsables de controlar las acciones de los robots. Dicho proceso evolutivo es combinado con aprendizaje durante la vida del individuo mediante reglas de aprendizaje Hebbiano generalizadas. Aparte del cambio entre tareas, otro subcampo destacablemente importante de la robótica colectiva es la comunicación entre robots. Combinado con la multitarea y usando los algoritmos previamente mencionados, el problema de la comunicación también es abordado en este Trabajo Fin de Máster. Es planteado mediante un experimento en el que solo un robot conoce la tarea que debe resolverse. Consecuentemente, dicho robot debe aprender a notificar la información correcta al resto del grupo. Para ello, proponemos un sistema de comunicación minimalista basado en la tecnología IR, que los robots pueden usar para interactuar. No obstante, la semántica de la comunicación tiene que emerger del proceso evolutivo. El tercer pilar de este Trabajo de Fin de Máster es la mejora y modificación radical de un simulador de robótica evolutiva desarrollado previamente por nosotros. Las mejoras más significativas son la incorporación de gráficos 3D, la detección de colisiones y la implementación de físicas realistas (usando la librería de python llamada pybullet). Asimismo, también se han implementado desde cero los algoritmos usados en las tareas de robótica colectiva. La estructura de este Trabajo de Fin de Máster consiste en dos experimentos tratando los temas de robótica colectiva del cambio de tarea y de la emergencia de comunicación. Después de la evolución, se muestra que los robots son capaces de resolver correctamente ambos experimentos con resultados muy destacables. Con el fin de respaldar los resultados, se ha usado una muestra estadísticamente significativa de 50 simulaciones independientes.

Palabras Clave— Robótica Colectiva, Computación Evolutiva, Redes Neuronales Recurrentes en Tiempo Continuo, Redes Neuronales Recurrentes, Comunicación Minimalista, Comunicación Emergente, Cambio de Tarea, MuliTarea, Neuroevolución de Topologías Aumentadas, Aprendizaje Hebbiano, Controlador Neuronal, Neuroevolución, Simulador de Robótica Evolutiva.

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 Acronyms

ANN: Artificial Neural Network

CTRNN: Continuous-Time Recurrent Neural Network

ER: Evolutionary Robotics

EA: Evolutionary Algorithm

FSM: Finite State Machine

GA: Genetic Algorithm

IR: Infrared

JSON: JavaScript Object Notation

MAML: Model-Agnostic Meta-Learning

MAS: Multi-Agent System

NAS: Neural Architecture Search

NEAT: NeuroEvolution of Augmenting Topologies

NES: Natural Evolution Strategy

RF: Radio Frequency

RNN: Recurrent Neural Network

SNES: Separable Natural Evolution Strategy

SR: Swarm Robotics

STDP: Spike-Timing-Dependent Plasticity

TWEANN: Topology and Weight Evolving Artificial Neural Networks

URDF: Unified Robot Description Format

xNES: Exponential Natural Evolution Strategy

Chapter 1

Introduction

This Master Thesis is framed within the intersection of the soft computing fields of collective robotics and multi-agent systems, evolutionary computation and artificial neural networks. Two experiments, composed by two different tasks each, are proposed and solved using groups of evolved, distributed and self-organized robots. The main aim is to study the emergence of communication, multitasking and task switching using evolutionary computation and artificial neural networks. More specifically, a type of recurrent neural network called continuos-time recurrent neural network is evolved using the combination of the neuroevolution of augmenting topologies algorithm and Hebbian learning rules. In the first experiment, the capabilities of evolutionary computation and recurrent neural networks to evolve robot controllers able to learn multiple tasks and switch among them at the correct timing is assessed. The proposed tasks are the pursuit of a mobile light source and the transportation of scattered solid cubes into a common nest area. The second experiment is devoted to the communication emergence using artificial evolution. The communication has to emerge in order to correctly solve the experiment because only one of the robots is actually aware of the task that has to be executed. Consequently, the robot that has access to this information has to communicate it to the rest of the group in order to accomplish the problem requirements. This chapter starts with an introduction to multi-agent robotics systems and a description of the state of the art relevant to this work. Subsequently, the different objectives and contributions of this Master Thesis are 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

The field of Multi-Agent Systems (MAS) is the research field that explores the use of multiple autonomous, distributed and intelligent entities that interact and collaborate in order to solve complex problems (see e.g. [1] or [2]). The applications addressed by MAS 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. Even though MAS systems can be also based on the competition among agents, this work is uniquely focused on cooperative multi-agent systems. MAS is a broad term that encompasses any problem in which many self-acting agents cooperate in order to address a common task. For instance, in [3], multi-agent systems have been used to construct Intrusion Detection Systems and, in [4], an anticipatory vehicle routing that is decentralized by means of delegate MAS is proposed. A recent breakthrough in the field of MAS have been presented in [5]. The authors merge competitive and cooperative multi-agent reinforcement learning by forming two teams of intelligent agents that learn to play the hide-and-seek game. They show how learning is conducted towards increasingly more complex agent behaviors, exposing the

emergence of unexpected and outstanding strategies. Nonetheless, in this Master Thesis, MAS concepts and techniques are applied to the field of robotics systems, which can be also denoted as collective robotics (see [6]). In collective robotics, multiple self-sufficient and simple robots are individually controlled in a distributed way in order to reach a common objective. A highly relevant and widely explored subfield of collective robotics is Swarm Robotics (SR), see [7]. In SR systems, there is a set of established conditions that a robotics system must fulfill in order to be considered a swarm robotics system. These requirements are the autonomy of individual robots, the inefficiency and simplicity of single agents, the homogeneity of the swarm, the existence of large swarm sizes and, lastly, the restriction to local inter robot communication and sensing of the environment. Moreover, scalability, robustness and flexibility are highly desirable properties in SR systems (see [8]). The fields of collective robotics and swarm robotics are widely inspired by how biological swarms work in nature [9], 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. The multi-agent systems designed and used in this Master Thesis are not strictly SR systems because the employed number of robots is remarkably scarce. However, the remaining requirements of SR are fulfilled and, more importantly, there are multiple concepts, techniques and insights used in our approach that are based on swarm robotics.

The summarized functioning of the self-sufficient robots commonly used in collective robotics is as follows. Firstly they are endowed with a set of sensors, which can be used to locally perceive the surroundings. Thereafter, the sensor measurements are processed by a controller program. Using the acquired information, the controller decides the actions that the robot has to perform in order to interact with the environment. Essentially, the controller defines the behavior and policy of the agents. Lastly, using a set of actuators, the planned actions are materialized and executed. The most important building block, and the one into which artificial intelligence enters into play, is the robot's controller. There are two main procedures used to build and design effective robot controllers (see [10]):

- *Behavior-based design methods*: are methods to design robot's controllers in which the researcher manually creates and handcrafts the entire controller functioning in response to the problem to be addressed. The main issue with this kind of controller design method is that it requires a great amount of time, effort and human knowledge to be accomplished. Moreover, there are problems whose vast complexity utterly hinders the manual creation of suitable solutions, regardless of the research knowledge and experience. Two commonly used behavior-based design methods are Finite State Machines (FSM), either deterministic or probabilistic, and artificial physics. In FSMs, the researcher defines a set of behavioral states of the robots and several stimuli dependent transition conditions among the states. In the case of Probabilistic FSMs, randomness is also included in the state transitions. Some examples of FSMs in SR are [11], [12, 13] or [14]. In contrast, artificial physics are mainly employed in tasks involving robot locomotion. There, the overall contribution of a set of virtual forces determines the direction of motion of agents. See, for instance, [15, 16, 17] for some examples of artificial physics.

- *Automatic design methods*: employs parametrized mathematical models whose parameters are fine-tuned using powerful optimization techniques. The optimization normally maximizes the estimate of the fitness function or long term reward with respect to the optimizable parameters within a constrained search space. Under the frame of automatic design methods, the research spotlight lies in both the mathematical model and the optimization algorithm. Due to its high versatility and computational power, the most popular and widely employed mathematical models are the Artificial Neural

Networks (ANN). ANNs without any kind of optimization mechanism are essentially empty containers. It is either learning or evolution what makes ANNs state of the art solutions in solving multi-agent robotics problems. The most important fields when referring to the optimization of ANNs in partially observable Markov decision processes and robotics are reinforcement learning (see e.g. [18, 19]) and evolutionary computation (see multiple examples below). These two fields are a noticeable example of the dichotomy of evolution and learning as different biologically plausible optimization mechanisms.

In this Master Thesis, evolutionary computation algorithms are used to evolve the parameters of a Recurrent Neural Network (RNN). Specifically, we utilize Continuous-Time Recurrent Neural Networks (CTRNN), see [20]. CTRNN are a greatly appropriate option because they generate actions not only based on the currently measured stimuli but also based on past experience and events. Essentially, CTRNN can provide short and medium term memory, which is critical in most robotics problems. Furthermore, in contrast to regular RNNs, CTRNNs operate in continuous time, making them particularly suitable for collective robotics. CTRNNs have been successfully used as robot controllers in multiple research works [21, 22, 23, 24, 25]. Moreover, the parameters (synapse weights and neuron parameters) are optimized using artificial evolution. Before continuing, let us briefly introduce few related terms that will be of much importance in the remaining of this document. Firstly, the application of evolutionary computation to the field of robotics is commonly denoted as Evolutionary Robotics (ER), see [26]. Additionally, the use of artificial evolution to tune the ANN parameters is known in the literature as Neuroevolution (see e.g. [27]). Moreover, in evolutionary robotics, the robot's controllers that are based on ANNs are referred to as neural controllers or neurocontrollers. There are multiple evolutionary computation algorithms that have been applied to the field of collective robotics over the last decades. Genetic Algorithms (GA) and Evolutionary Algorithms (EA), as the most basic evolutionary computation processes, have been traditionally employed in multiple occasions [21, 22, 23, 24]. In [21], the neural controllers, based on CTRNNs, are evolved using a generational EA with the objective of reaching the emergence of a flocking behavior of the robots. Besides, the authors of [23] use a canonical GA to evolve alignment behaviors of robots, so that the heading orientation of the entire swarm converges to the same direction. The robots are also controlled by CTRNN models. In [22], the target problem faced using the combination of an EA and CTRNN controllers is a foraging task. Furthermore, they designed a foraging problem in which robots must accomplish a role switching in order to correctly complete the requirements. In contrast, in [24], a CTRNN and a generational EA are harnessed in order to solve the cooperative transportation of heavy objects. Aside from GA and EA, other artificial evolution algorithms has been applied to the collective robotics. Natural Evolution Strategies (NES), see [28], have been employed in [25] to solve the tasks of leader selection and swarm borderline identification by evolving the parameters of a CTRNN. Furthermore, NeuroEvolution of Augmenting Topologies (NEAT) has been used in [29]. There, the authors solve an aggregation task of a swarm of simulated robots using NEAT boosted with novelty search. They show that this combination leads to an increase of the population's diversity, an improvement of the convergence speed and a decrease in topological complexity of the NEAT solutions. In this Master Thesis, the evolutionary algorithm that is used to evolve the CTRNN models in the proposed experiments is NEAT. In this way, we leverage the NEAT capabilities of not only evolving the ANN parameters but also optimizing the network topology (see Chapter 2 for a detailed explanation of NEAT functioning). In our work, we combine NEAT evolution with a lifetime learning of the robots using Hebbian learning rules (see Chapter 2). The tandem of evolution and lifetime learning drastically enhances the convergence speed and smooths the search space, easing the access to areas with fitter solutions. Under the frame of collective robotics and as far as we are concerned, Hebbian learning, combined with evolution

processes, has not been explored in the literature.

Another important aspect of collective and swarm robotics to be briefly highlighted is the set of tasks that are commonly faced in the literature (see [30, 10, 31]). Among the most treated problems, those that outstand are the tasks of foraging (see [32, 33, 34, 11, 35, 14, 36]), flocking (see [21, 37, 15, 16]), swarm aggregation (see [38, 12, 39, 40]), role allocation (see [41, 42, 43, 44, 45, 46]) and object transportation (see [47, 48, 49]). Other less common, albeit also significative, applications of collective robotics are swarm leader election (see e.g. [25]), orientation consensus (see e.g. [23]) or shape formation (see e.g. [50]), just to mention few of them. These papers, and most of those available in the literature, are generally devoted to the resolution of a single task. Nonetheless, a remarkably challenging and interesting topic is the multitasking in collective robotics. Multitasking means that swarm members simultaneously learn to solve multiple, different and independent tasks. More precisely, it is of special interest for this Master Thesis the multitasking problem of task switching. In task switching, the robots not only have to learn different tasks using the same model but they also have to be able to change, at runtime, the task to be solved at the precise timing. Even though it is a sequential multitasking, robots still have to learn all the tasks. In this Master Thesis, task switching is one of the main pillars and topics to be studied under the frame of collective robotics, evolutionary computation and artificial neural networks. This means that the knowledge about how to solve multiple tasks and switch among them has to emerge from and artificial evolution process and be embedded into a CTRNN model. We hypothesize and, afterwards, observe that depending on the task to be addressed at each time instant, the dynamics of the evolved CTRNNs are driven towards a different trajectory or area within the state space of the network. The switch among multiple tasks has been traditionally accomplished using either threshold-based methods (see e.g. [51, 52]) or probabilistic methods (see e.g. [14, 53]). However, these procedures are not significantly relevant for this Master Thesis because we study the task switching problem from an evolutionary point of view. Several authors have addressed the problem of task switching and multitasking from the perspective of evolutionary computation and ANNs. In [54], the author evolves a multilayer perceptron to learn to correctly switch among three different tasks. The proposed tasks are to closely follow a target robot, to collect objects and to explore the environment. The evolution is accomplished using multiobjective evolutionary computation, so that the sought outcomes of the evolution are no longer point solutions but a curve of optimal solutions. This curve is called Pareto front and its elements provide the optimum balance in maximizing the fitness of all the tasks simultaneously. A similar approach is considered in [55], where the tasks to be commuted by the robot are approaching to a sound source and reaching several lights distributed along the environment. Tuci et al. address in [22] a task switching problem in a foraging experimental setup. The members of a team of 5 robots have to learn to distribute roles of patrolling the nest area and exploring and seeking food outside the nest, and perform the task switching at the precise timing. They show that CTRNN controllers evolved using an EA can successfully accomplish the experiment. Even though it is not strictly related with the topics treated here, the algorithm of Model-Agnostic Meta-Learning (MAML), presented in [56], has to be mentioned because of its relevance to the multitasking and task generalization problems in artificial intelligence. MAML proposes a general and versatile algorithm that finds a solution that is useful to multiple tasks, instead of fully overfitting to one of them. Thereafter, when a precise task is presented, a few shot learning process is carried out with a small number of gradient descents in order to solve it. MAML is model and task agnostic, meaning that it can be used with any model and to any machine learning problem (supervised, unsupervised or reinforcement learning).

In addition to task switching, the second pillar of this Master Thesis is the emergence of communication among robots using evolutionary computation. Furthermore, we explore the

limits of task switching by combining it with direct communication across the swarm. More precisely, we impose that only one robot is actually aware of the task that has to be executed at each time step and, thus, there must be a communication phase before the task switching is accomplished. In collective robotics, the swarm members can interact locally and with a remarkably restricted communication system and means. The most common technologies used in real collective robotics to cope with the swarm communication are IR [24, 29, 57, 58, 59, 25, 23], RF [60, 61, 62] and sound [63]. IR is commonly used in robotics setups with short range communications [64]. Even though it involves several disadvantages, such as ambient light distortion, interferences, communication death zones, impossibility to send and receive at the same time or low data rates (see [57, 65]), it is very suitable for restricted collective robotics due to its simplicity, cheapness and low power consumption. Moreover, unlike RF or sound technologies it allows directionality awareness in the communication. In this Master Thesis, the communication system designed and implemented is based on IR technologies, albeit we do not address the experiments with real robots. We implemented some of the drawbacks and limitations of IR communications within the simulations. For instance, the communication range is relatively low, there is a exponentially decaying distance and misalignment attenuations of the received signal and, very importantly, the communication receivers can only perceive one message at each simulation cycle. For a detailed explanation of this features, among other functioning characteristics of the communication system, see Chapter 3. Aside from the technological issues that allow communication in the first place, according to [66], the communication among robots can be:

- *Stigmergy* (see e.g. [34, 35]): robots indirectly communicate by using the environment as a medium. Normally, this kind of communication is inspired by how ants interact in nature using pheromones.

- *Interaction via the state* (see e.g. [21, 37, 24, 29]): robots indirectly interact by sensing each other throughout their sensors. A clear example of this type of communication is the perception of the approximate distance to solid objects, such as other robots, using IR distance sensors.

- *Direct communication* (see e.g. [63, 25, 67, 23, 68]): robots can communicate directly and explicitly by the exchange of messages or information among them. Direct communication is a broad term that can refer to multiple communication mechanics such as signalling processes, complex language semantics or exchanges of the measured sensors' states.

Under the frame of our work, the most relevant type of communication is the direct communication, albeit the interaction via the state can also play an important role in the experiments. Moreover, it is important to keep in mind that the researcher is only responsible of designing and defining the communication mechanics, limitations and capabilities. In essence, the researchers define the rules of the communication game. Nonetheless, it is a labor of the evolution process to decide how robots leverage the communication means at their disposal. In these cases, it is said that the communication semantics emerge as a result of evolution and learning. After this remark, there are two main types of emergent communications according to [69]: (**1**) *Abstract communication* (see [67, 63, 70, 68, 25]), in which the communication semantics fully lie on the message being transmitted between robots. The message content is the only part of the communication that is communicative and the robots do not use any information about the context of the message within the environment. (**2**) *Situated communication* (see [23, 71, 25]), in which both the message content and the underlying environmental context carry information. In other words, even though the message is informative, it cannot be understood without its contextualization. Some example of context information are the estimate of the distance to the sender robot or the

relative orientation from where the message was received. In [67], the authors exhaustively study the emergence of communication with an EA applied to an ANN and with an initially non-communicative group of robots. They also analyze how the communication can be progressively complexified as evolution advances. After evolution, they expose that multiple signalling semantics arise in different states of the transmitting robot within the environment. The authors of [63] proposed an experiment in which two robots have to cooperate in order to move a revolving door. The force of both robots is required in order to correctly rotate the door and the rotation sense depends on the state of some arena lights and ground areas. Communication between robots is required because each agent only knows part of the environment information needed to successfully rotate the revolving door. Thereafter, the authors propose a very simple sound based signalling communication that the robots can use to coordinate and exchange information. They use evolutionary computation and CTRNNs to verify that the signalling communication can emerge using such a simple experimental setup. Gutierrez et al. [23] address the task of heading alignment in a swarm of mobile robots using a generational GA and CTRNNs. Their results show that both a 3 bit message content and the orientation of the message sender are relevant in the emerged communication. In [25], it was verified that the same neural controller, optimized through artificial evolution, can lead to either an abstract or a situated communication, depending on the task to be solved. This experiment was carried out using CTRNNs and NES algorithms, applied to the tasks of leader election and swarm borderline identification. An abstract communication emerged in the case of the leader election, while the borderline identification led to the emergence of purely situated semantics.

1.2 Objectives and Contributions

In this Master Thesis, we addressed the challenging topics of communication emergence and task switching under the frame of collective robotics and artificial evolution. Our work is structured in two experiments, coping with each of the mentioned problems. The first experiment is devoted to the resolution of two tasks that have to be carried out by all the robots sequentially. Robots not only have to learn to solve both tasks but also to swap between them at the correct timing (task switching). The second problem assesses the emergence of communication through evolution. Only one robot is aware of the task to be executed and, thus, it has to notify it to the rest of the group. In both experiments, CTRNNs are evolved using NEAT and developed by means of Hebbian learning. In the light of this brief explanation, one of the main objectives of this Master Thesis was the successful resolution of both experiments. Aside from the sought positive results, the goal was to response to the following questions:

- Is it possible to evolve neural controllers in order to learn two utterly different tasks simultaneously? If the answer is positive, how does the neural network manage to switch among behavioral states?
- Provided that the former question is correctly validated, are CTRNN neural controllers capable of successfully learning when to switch among the behavioral regimes corresponding to the tasks?
- Is it possible to merge the previously studied task switching with communication among robots? What happens if only one robot of the group knows the task to be executed and the rest must fully rely on it? Can evolution lead to emergence of a sufficiently precise inter-robot communication required to tackle with this scenario?

Moreover, provided that the results obtained in the experiments lead to favorable responses to the previous questions, an important aim of this Master Thesis was to deeply analyze

the emerged communication and the task switching behaviors. In contrast to the mentioned objectives, several equally important milestones are related to the development of a software simulator devoted to the design, execution, optimization and analysis of evolutionary collective robotics systems. The aim was to drastically improve a previously existing robotics simulator written in Python programming language and developed by us in previous projects and thesis [72]. Specifically, the precise and minimal objectives to be obtained were the addition of 3D renderings, the implementation of realistic physics simulations and solid object collisions and the programming of the algorithms used in the experiments (NEAT and Hebbian learning). All these goals, among other collateral code upgrades, were successfully achieved. Clearly, all these software improvements are fulfilled firstly within the Master Thesis duration, because they are used in the proposed experiments.

Once the initial Master Thesis objectives have been established, the contributions can be listed and compared to the imposed goals. Aside from the software simulator objectives, which were proficiently implemented and can potentially contribute to future projects and research activities, the rest of the contributions are enlisted below:

- The task switching experiment is correctly solved, showing that the evolved robots can proficiently address both tasks and swap between them at the precise time instant. The results are supported by graphics showing the robots' behavior and using an statically significant sample of independent simulations.

- The communication emergence is achieved in the second experiment. The robot that is aware of the correct task is able to notify it to the other robots. Similarly, the rest of the group is able to process the received message and address the targeted task in response. The emerged communication, which is essentially a unidirectional signalling process, is post analyzed using 50 independent simulation trials.

- In order to tackle the previous contribution, a minimal IR-based communication system, that the robots can use to interact, is designed and simulated.

- In response to the post analysis, we hypothesize that the swap among tasks corresponds to a drastic change of the dynamics of the CTRNNs, driving the state space trajectories towards different areas. Event though this fact was observed in the post analyses, a deeper research has to be accomplished in order to reliably affirm and validate this statement.

- CTRNN models, NEAT algorithm and Hebbian learning are implemented and jointly used to successfully solve the experiments.

1.3 Document Layout

The remaining of this document is structured as follows. Firstly, Chapter 2 is devoted to explain the theoretical background that will be required in the subsequent chapters. Specifically, the chapter starts with a description of Continuous-Time Recurrent Neural Networks, the type of recurrent neural network used in the experiments of this Master Thesis. The overview is provided starting from the neuron level, specifying the dynamics of individual point neurons using firing rate models. Subsequently, the CTRNN is explained on top of the neuron's definition. The CTRNN is viewed from two complementary perspectives, which are as a weighted directed graph of neurons interconnected through synapses and as a non linear dynamical system. Thereafter, NEAT is presented as the evolutionary computation algorithm used to optimize the CTRNN model. We exhaustively describe the main parts of NEAT, which are the mutation, selection and crossover operators, the functioning of the

historical markings and the speciation process. To conclude with the theoretical preliminaries, Hebbian learning rules are outlined as the lifetime learning procedure of CTRNNs.

Chapter 3 presents the evolutionary swarm robotics simulator developed and used along this Master Thesis. It starts with an overview of the software, emphasizing the environment were the experiments are carried out, the physics engine that is leveraged in order to create realistic physics, collisions and 3D renderings and the configuration files, that fully describe the experimental setups. Moreover, the ANNs and evolutionary computation processes are treated from an implementation perspective. Thereafter, the functioning and implementation of the sensors, actuators and communication system, that the swarm agents can harness, is deeply explained.

Additionally, the experiments that are carried out in this Master Thesis are defined in Chapter 4 using the simulator presented in Chapter 3. For each of the two experiments, which are devoted to the study of task switching and communication, the principal aspects of the experimental setup are exposed. Specifically, the chapter starts with a detailed description of the experiment definition and mechanics, highlighting the tasks that the robots have to accomplish and the goal to be achieved. Thereafter, the fitness functions, that are used to evaluate how proficiently the NEAT solutions behave in the tasks, are displayed and justified. The neural controllers are also shown in this chapter. Owing to the fact that NEAT also evolves the ANN topology, only the initial CTRNN architectures are illustrated at this point. Chapter 4 ends with the presentation of the evolutionary experimental setup, gathering the best found hyperparameters and the bounds of the search space.

After the evolution process, the results provided by the best performing genotypes of each experiment are displayed. In order to construct the graphics that serve as the demonstration of the robot's behavior, we collect an statically significant sample of 50 independent simulations. Consequently, the gathered sample is used to showcase statistically reliable results.

Chapter 6 concludes this Master Thesis, highlighting the most relevant results. 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. 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 three differentiated topics. Firstly, a detailed description of 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 overall neural network is explained from two coexisting perspectives, namely, as a directed graph and as a non-linear dynamical system. Secondly, the functioning and theory of the NeuroEvolution of Augmenting Topologies algorithm (NEAT) is presented as the evolutionary algorithm used to optimize the artificial neural networks previously defined. NEAT allows not only the optimization of the synapse weights but also the evolution of the neural network topology. Special attention is paid to the main topics of NEAT algorithm, which are the historical traces of genes and how the crossover operator uses them to produce suitable offspring, the architectural mutation operators and the speciation process to preserve innovation. Lastly, evolution is combined with lifetime learning as a meta-learning process. Specifically, individuals learn from interaction and experience by means of synapses' Hebbian learning rules. These rules update the connection weights in response to the measured stimuli and the current state of the network. We use the generalized ABCD Hebbian learning rule, whose rule parameters are also optimized with the NEAT algorithm.

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 [73] and its variants [74, 75], Continuous-Time Recurrent Neural Networks (CTRNNs) [20] 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: as a directed graph and as a dynamical system.

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 [76]) 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 [77, 76] 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 [77] or [78].

Assuming a 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 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 respectively. The weights of the edges are attached to each connection arrow and the ordering of the neurons is inside the vertices.

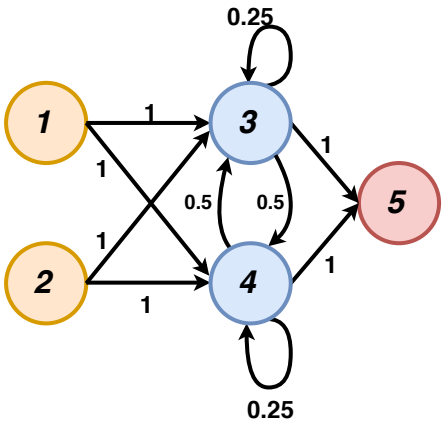


Figure 2.1: Example of a simple CTRNN with 2 inputs (yellow), 2 hidden neurons (blue) and 1 motor neuron (red).

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:

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

Finally, in order to iteratively solve the system of differential equations in Eq. 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 $\mathbf{v}(0) = (0, \ldots, 0)^\top$.

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

2.2 NeuroEvolution of Augmenting Topologies

Neuroevolution is the application of evolutionary computation algorithms to optimize the parameters of artificial neural networks (see [27]), leading to large dimensional optimization problems. There are multiple variants of neuroevolutionary algorithms, depending on the evolutionary mechanics, the genotype representation and encoding employed or the artificial neural network model to be evolved. Just to mention some of them, in [38, 21, 79, 80, 22, 23] ANNs are evolved using genetic or evolutionary algorithms. Cooperative coevolutionary algorithms are proposed in [81, 82] as optimizers of ANNs leading to multiple subpopulations of neural parameters being evolved in parallel. Evolution strategies [83, 84], which directly optimize a parametrized search distribution from where genotypes are sampled, have been also explored in the context of ANN optimization. All the mentioned neuroevolution algorithms optimize the parameters of artificial neural networks with fixed architectures or topologies.

This means that the researcher is responsible of deciding and designing the ANN architecture (number of layers and neurons or how neurons are connected). In most cases, the resulting neural structure has either insufficient computational capabilities or an excess of complexity and degrees of freedom. In the former scenario, the optimization algorithms may be unable to reach acceptable solutions to the requested problem. In the latter case, the search space of the optimization problem becomes unnecessarily large dimensional, hindering and slowing down the evolution process. In order to deal with this problem, Neural Architecture Search (NAS) algorithms optimize not only the neural network parameters but also the architecture of the ANN. The term NAS refers to the search of ANN topologies generally comprising also other non-evolutionary families of ANN optimization algorithms, such as reinforcement learning. Focusing on evolutionary computation, the family of algorithms designed to evolve the ANN topologies has been traditionally known as Topology and Weight Evolving Artificial Neural Networks (TWEANN). In this Master Thesis, the employed TWEANN algorithm is the NeuroEvolution of Augmenting Topologies (NEAT) algorithm, which is one of the most widely used and established algorithms in the field. Although there are several innovations and improvements of NEAT proposed in the literature, such as HyperNEAT [85], CoDeepNEAT [86] or Weight Agnostic Neural Networks [87], in this Master Thesis the original proposal of NEAT as described in [88] is used. NEAT is a type of genetic algorithm that simultaneously optimizes the architecture of the ANN and its weights.

A population of candidate solutions, namely individuals, genotypes or chromosomes, is 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 of the evolution process. 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). Owing to the fact that NEAT genes are not only scalar parameters but also contain topological information, the genetic encoding of the genotypes $\mathscr{G}_i$ in the population is not a fixed length direct encoding as in most neuroevolution problems. Instead, a genotype is composed by two sets of genes, namely, a list with the node genes and a list with the connection genes. Each node gene, representing a neuron in the CTRNN, contains the corresponding neuron parameters, such as the membrane time constant, the neuron bias and the neuron gain. Alternatively, the connection genes register the presynaptic and postsynaptic neurons and the connection weight, among other possible synapse parameters.

Besides, the fitness function ($f(\mathscr{G})$) used to evaluate genotypes depends on each particular application and will be exposed in Chapter 4. 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 $g = 0$. All the individuals in the initial population start with the same minimal neural architecture. Normally, the initial topology is a fully connected ANN with no hidden layers between input and motor layers. However, if the number of input nodes is remarkably large, it is advisable to include an initial hidden layer with few neurons in order to reduce the number of connections. On the contrary, the weights and the rest of parameters are randomly initialized according to a gaussian distribution $\mathcal{N}(0, 0.1^2)$. In our case, the unique exception is the initialization of neuron time constants, in which a Rayleigh distribution with $\sigma = 0.7$ is applied.

Fig. 2.2 depicts the data flow of NEAT, highlighting its main stages in each generation. These evolutionary stages are the fitness evaluation, parents selection, recombination or

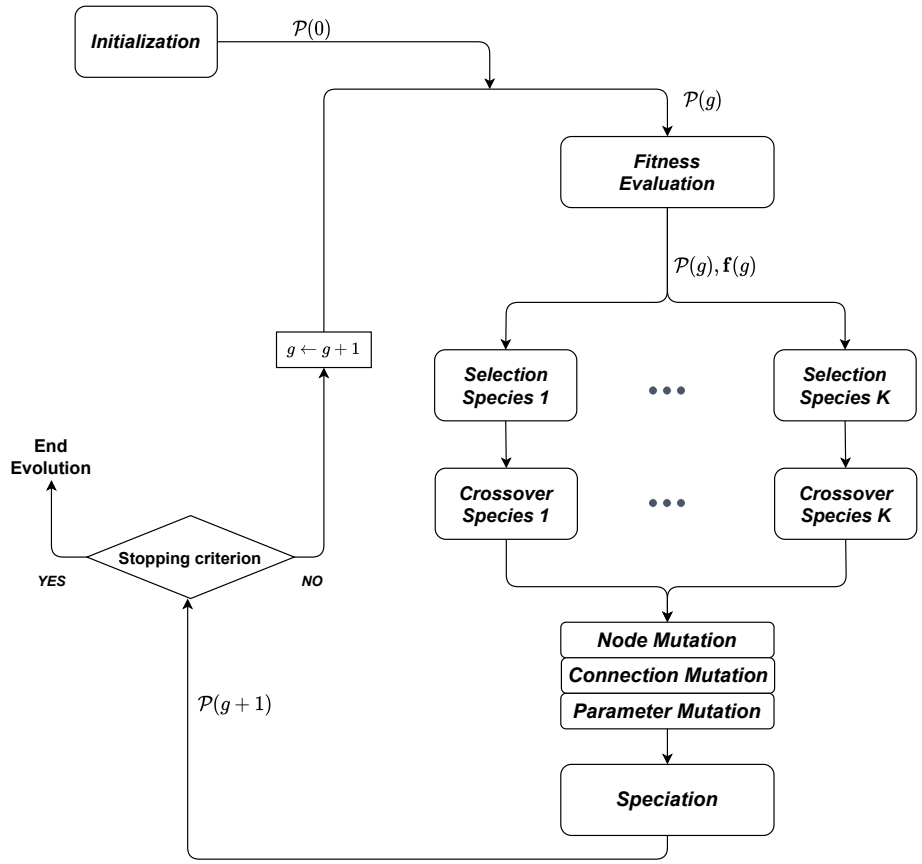


Figure 2.2: Diagram summarizing the data flow of NEAT algorithm.

crossover, mutation processes and speciation. The stopping criterion is either fulfilled by reaching some predefined population mean fitness or when a maximum number of generations are elapsed. These steps will be described in detail in the remaining part of this section. Nonetheless, as a first introductory approach, they can be summarized as follows:

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 a new individual.

3. *Mutation*: The offspring genotypes, resulting from the recombination phase, are subject to a mutation step that, with some mutation probability, 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. In NEAT, there are three types of mutation operators: the parameter mutation, the node mutation and the connection mutation.

4. *Speciation*: In order to protect architectural innovation resulting from recent mutations in the population, NEAT uses speciation techniques. Speciation partitions the

population into isolated niches or species so that the members of each species compete and recombine only with their neighbors. With speciation, the selection and the crossover operators are applied to each species separately.

2.2.1 Mutation operators and innovation numbers

Mutation operators, present in most evolutionary computation algorithms, are responsible for searching new areas of the search space in order to find more suitable solutions and avoid early convergence to local optima. Mutations are generally applied to genotypes or genes (depending on the precise operator) with a very small probability and alter the gene or some part of the genome, producing an slightly different and hopefully better individual. Those mutation that lead, in the short term, to a fitness rise will likely be preserved. On the contrary, mutations resulting in a performance decrease will eventually vanish away from the population. In NEAT, there are three different mutation operators that are applied to each genotype. In brief, and before entering into details, their roles are to add new connections, add new neurons and slightly modify the weights (or parameters in general) of the neural network. More specifically the functioning are role of each mutation operator is provided below:

- **Connection Mutation**: For each genotype, and with a small probability $p_{m,c}$, a new connection is created in the CTRNN architecture. The presynaptic and postsynaptic neurons of the new connection are randomly selected among those pairs of neurons with no existing synapse between them. In our case, the self connections are allowed. The weight and the rest of the synapse parameters (e.g. the learning rule weights, see Section 2.3) of the connection are sampled randomly from $\mathcal{N}(0, 0.1^2)$. Fig. 2.3 shows an example of a connection mutation in which a new synapse is added to the network.

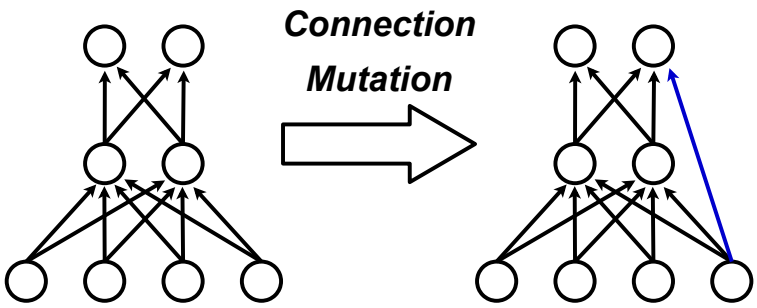


Figure 2.3: Example of a NEAT connection mutation.

- **Node Mutation**: aside from connection mutations, the artificial neural network can also grow and explore new topologies by means of neuron or node mutations. With a low probability $p_{m,n}$ (commonly lower than $p_{m,c}$) the node mutation functions as follows. Firstly, an existing connection is randomly selected in the ANN. The chosen synapse connecting, say, neurons A and B, is disabled so that it becomes dysfunctional until recombination operations enable the gene again. The new node resulting from the mutation replaces the disabled connection and two new synapses are also created. One of the included synapses connects node A with the new node while the other one connects the new node with neuron B. Therefore, the mutation avoids creating isolated nodes in the ANN.

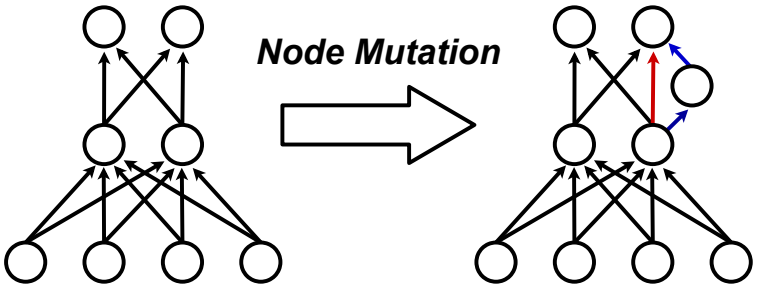


Figure 2.4: Example of a NEAT node mutation. The blue arrows denote the new synapses connecting the new node with the rest of the ANN. The red arrow represents the disabled connection resulting from the mutation.

Fig. 2.4 shows an example of a node mutation, where the explained mechanics can be observed. The red arrow represents the disabled connection and the blue arrows represent the synapses added in order to connect the new node with the rest of the ANN. The mutated node is the circle between the blue connections. The weight and the rest of the parameters of the connection joining the new node with B have the same value as those in the disabled connection. On the contrary, the synapse connecting neuron A with the mutated node has a random weight sampled from $\mathcal{N}(0, 0.1^2)$. The parameters of the new neuron are also selected randomly. More precisely the neuron gain g and the bias β are sampled from $\mathcal{N}(1, 0.05^2)$ and $\mathcal{N}(0, 0.1^2)$ respectively. The neuron time constant τ is sampled from a Rayleigh distribution with a $\sigma = 0.7$.

- **Parameter Mutation**: Lastly, with a probability $p_{m,w}$, the parameters of each connection and node genes are slightly altered. As in most real coded evolutionary algorithms, the genes are modified using the gaussian mutation, in which the gene parameter is resampled from a gaussian distribution with the mean located at the parameter current value and a small variance.

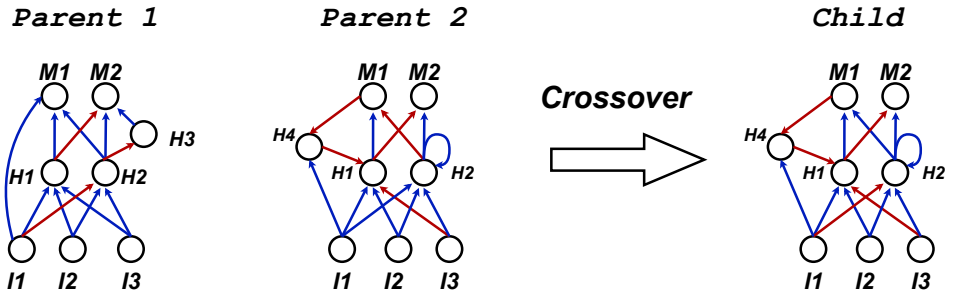
$$\theta_m = \theta + \epsilon, \quad \epsilon \sim \mathcal{N}(0, \sigma_w^2) \tag{2.10}$$

Eq. 2.10 shows the gaussian mutation operation for a single gene parameter θ. θ can be either a connection weight, a neuron time constant, a neuron bias or a neuron gain. Additionally, the learning rule parameters of the synapses are also subject to the same mutation.

In order to avoid the competing conventions problem (see [89]) and to allow a functional recombination between different sized genotypes, NEAT implements historical markings of genes. Specifically, an innovation number is assigned to every gene when it is created (either by mutation or initialization). Innovation numbers are traces of a particular topological trait of the gene, meaning that two genes with the same innovation number correspond to the same ANN structural information (e.g. the same connection or the same neuron). Nonetheless, the value of the weight or the other gene parameters are not tracked by the historical marking and may be different. There is a global innovation pointer which indicates the most recent innovation number assigned to a gene. Every time that a mutation occurs and a connection is created, the global innovation pointer is increased by 1 and assigned as innovation number to the new gene. Every gene is also registered and stored with its corresponding innovation number along the entire evolution. Thus, in case that a mutation results in a connection that has already occurred in the evolution, the innovation number corresponds to the value previously registered.

2.2.2 Crossover operator

NEAT algorithm implements a crossover operator that suitably allows the inheritance of both weights, neuron parameters and topology of the parent chromosomes. It leverages the innovation historical traces of genes in order to match and recombine genotypes of different sizes and represent diverging ANN architectures. Specifically, genes that are present in both parents are identified according to their innovation numbers. The shared genes of the parents, both connection and neuron genes, are transmitted to the offspring with the same innovation number. The parameters of the gene (for example the weight in the connection genes) are inherited randomly from one of the parents with equal probability. Additionally, if the inherited connection gene is disabled in any of the parents, there is a probability of 0.75 (as proposed by the authors of NEAT) that the connection is also disabled in the child genotype. Alternatively, those genes that are only available in one of the parents are inherited from the fittest parent. In order to clarify the crossover mechanics, Fig. 2.5 illustrates a toy example of



(a)

P1	Innov=1 Pre=I1 Post=H1 w=0.5	Innov=2 Pre=I1 Post=H2 w = -0.2	Innov=3 Pre=I2 Post=H1 w=2	Innov=4 Pre=I2 Post=H2 w=0.15	Innov=5 Pre=I3 Post=H1 w=0.3	Innov=6 Pre=I3 Post=H2 w=2	Innov=7 Pre=H2 Post=H3 w=-0.35				Innov=11 Pre=H1 Post=M1 w=0.65	Innov=12 Pre=H1 Post=M2 w=-0.01	Innov=13 Pre=H2 Post=M1 w=-0.8	Innov=14 Pre=H2 Post=M2 w=0.12	Innov=15 Pre=H3 Post=M2 w=1.05		Innov=17 Pre=I1 Post=M1 w=3
P2	Innov=1 Pre=I1 Post=H1 w=0.2	Innov=2 Pre=I1 Post=H2 w=1	Innov=3 Pre=I2 Post=H1 w=0.1	Innov=4 Pre=I2 Post=H2 w=0.25	Innov=5 Pre=I3 Post=H1 w=-1	Innov=6 Pre=I3 Post=H2 w=3		Innov=8 Pre=H2 Post=H2 w=0.05	Innov=9 Pre=I1 Post=H4 w=0.67	Innov=10 Pre=H4 Post=H1 w=-0.2	Innov=11 Pre=H1 Post=M1 w=1.4	Innov=12 Pre=H1 Post=M2 w=-0.6	Innov=13 Pre=H2 Post=M1 w=-1.5	Innov=14 Pre=H2 Post=M2 w=1.2		Innov=16 Pre=M1 Post=H4 w=-0.85	
CHILD	Innov=1 Pre=I1 Post=H1 w=0.5	Innov=2 Pre=I1 Post=H2 w=-0.2	Innov=3 Pre=I2 Post=H1 w=0.1	Innov=4 Pre=I2 Post=H2 w=0.15	Innov=5 Pre=I3 Post=H1 w=-1	Innov=6 Pre=I3 Post=H2 w=3		Innov=8 Pre=H2 Post=H2 w=0.05	Innov=9 Pre=I1 Post=H4 w=0.67	Innov=10 Pre=H4 Post=H1 w=-0.2	Innov=11 Pre=H1 Post=M1 w=0.65	Innov=12 Pre=H1 Post=M2 w=-0.6	Innov=13 Pre=H2 Post=M1 w=-0.8	Innov=14 Pre=H2 Post=M2 w=0.12		Innov=16 Pre=M1 Post=H4 w=-0.85	

(b)

Figure 2.5: Example of a connection crossover. (a) Crossover in the phenotype domain (ANN graphs). The blue connections represent excitatory synapses and the red connections highlight the inhibitory connections. (b) Crossover in the genotype domain (list of connection genes). Innov is the innovation number of the gene, pre and post are the names of the presynaptic and postsynaptic neurons and w is the weight of the connection.

a connection recombination between two ANNs. In Fig. 2.5a, the crossover is depicted in the phenotype domain, highlighting the excitatory connections in blue and the inhibitory synapses in red. Supposing that the parent 2 has a fitness score larger that parent 1, the recombined child results in the same architecture as parent 2. However, it can be observed that the weights of the child phenotype are a random mixture between the weights of both parents. Fig. 2.5b shows the same crossover example but in the genotype domain (list of connection genes). Each gene shows the presynaptic and postsynaptic neurons, the innovation number

and the weight. Other gene variables, such as the bit stating if the connection is enabled are omitted for simplicity. In this scenario, it can be clearly observed that disjoint and excess genes are inherited only from the fittest parent (P2) while the rest of the weight parameters are transmitted randomly.

2.2.3 Speciation

As one of the main pillars of NEAT, speciation is included in the algorithm. Speciation or niching [90] partitions the population into clusters called species. The genotypes in each species share properties or genetic traits with the other members of the niche. Thereafter, recombination operation is only applied among individuals in each species in an isolated way. Even though speciation implementations can have a very low interspecies reproduction probability, in the case of this Master Thesis the crossover between individuals of different niches is not allowed. Furthermore, individuals only compete with other members of the species in the selection process. The main role and motivation of using speciation in evolutionary algorithms is to tackle multimodal and multiobjective optimization problems, in which, each niche tends to a different local optima. Nonetheless, the authors of NEAT propose a different function of speciation. Essentially, speciation is also used for maintaining and protecting innovation after structural mutations. Specifically, in absence of niching, a node or connection mutation would rarely lead to an immediate fitness rise because the weights or neuron parameters still need to be developed and fine-tuned.

One of the most critical steps when designing evolutionary algorithms with speciation is the definition of a similarity distance between genotypes. Depending on the problem and evolutionary algorithm used, the criterion for computing the similarity between two genotypes can be:

- Search space: the similarity distance between individuals is based on how different their genes are. The distance between genomes can be, for instance, the Euclidean distance or the Manhattan distance in the case of the real coded genetic algorithms or the Hamming distance in the binary coded genetic algorithms.
- Behavioral space: the similarity distance between individuals is based on how differently their phenotypes behave in the problem to be solved. The behavioral distance depends on the precise task or problem to be solved.

In NEAT, the similarity metric compares the genes of the individuals (search space distance). However, owing to the fact that NEAT genes represent architectural traits of a CTRNN, the distance is different from the previously mentioned metrics. More precisely, the similarity distance of NEAT speciation is shown in Eq. 2.11.

$$\delta(\mathscr{G}_1, \mathscr{G}_2) = c_1\, E + c_2\, D + c_3\, \frac{1}{d_c}\, \|\boldsymbol{\theta}_1 - \boldsymbol{\theta}_2\|_1 \tag{2.11}$$

where E and D are the number of excess and disjoint genes (genes that are only present in one of the parents) between genotypes $\mathscr{G}_1$ and $\mathscr{G}_2$. $\boldsymbol{\theta}_1$ and $\boldsymbol{\theta}_2$ are vectors gathering all the gene parameters (weights, time constants, neuron biases, ...) from the genes that are common to both genotypes $\mathscr{G}_1$ and $\mathscr{G}_2$. Furthermore, d_c represents the number of common genes and, thus, the dimension of $\boldsymbol{\theta}_1$ and $\boldsymbol{\theta}_2$. In addition, c_1, c_2 and c_3 are hyperparameters that indicate the importance of each term when computing the similarity distance. Besides, a similarity threshold δ_t is defined, so that if the distance between two individuals is larger that δ_t then they are dissimilar enough to belong to different species. Moreover, each species has a representative genotype that is used to compute the distance of individuals to the niche. This means that a genotype is assigned to a species if its distance to the species representative is lower than δ_t. In a scenario with an individual close to multiple species, the selected

niche would be the one to which the distance is lowest. Consequently, the speciation process of NEAT is composed by the following steps: (i) For each genotype, the distance to the representative of each existing species is computed according to Eq. 2.11. (ii) The genotype is assigned to the nearest species. If the individual is not close enough (distance lower than δ_t) to any existing species, then a new species is created with the genotype as representative and unique niche member. (iii) After allotting a niche to every population individual, a new representative is chosen for the next generation as the nearest species genotype to the prior representative. (iv) Potential extinction of species is verified. A species is considered extinct if there is no organism of the current population belonging to it.

Additionally, the crossover stage described in Section 2.2.2 is applied to each species separately. This implies that genotypes can only recombine with other members of its own niche. Furthermore, the amount of offspring alloted to each species is dynamic and depends of the mean fitness of the niche members (notice that the mean fitness of the species members equals the sum of explicit adjusted fitnesses computed as in [88]). The number of allowed offspring of each niche is computed as in Eq. 2.12.

$$n_{off,i} = \left\lfloor \frac{\overline{F}_i}{F_{tot}} \lambda \right\rfloor \tag{2.12}$$

where $\overline{F}_i$ is the mean fitness of the i-th species, F_{tot} is the total fitness resulting from the sum of the fitness scores of all the individuals in the population and λ is the population size.

2.2.4 Selection operator

The selection operator is the evolutionary process in which a subset of the population individuals is selected as parents used by the crossover algorithm to produce the offspring. The criterium employed to chose genotypes is mostly based on their achieved fitness scores, preserving the fittest individuals while filtering out the worst ones. In the case of NEAT, the selection operator is applied to each species separately. Therefore, organisms compete with their own niche neighbors in the selection process.

In constrast to the NEAT original paper [88], in which the authors use truncation selection, tournament selection is used as selection operator in this Master Thesis (see [91]). Let $\mu \in \{1, \ldots, \lambda\}$ be the number of genotypes to be selected as parents. In tournament selection, 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.

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 as it reduces selective pressure (if tournament size K is sufficiently small).

2.3 Hebbian Learning

Adaptive ANNs (or plastic ANNs), see e.g. [92], are a kind of neural models that can dynamically adapt their parameters and learn from experience at runtime by interacting with the environment. More precisely, the weights of the synapses can be slightly raised or decreased according to the current state of the ANN. One of the mains fields studying adaptation and plasticity in neural networks is Hebbian Theory [93], that is summarized by the following quote: *"Neurons that fire together wire together"*. Specifically, if two neurons are producing action potentials simultaneously or in a short time window, their activation purpose is likely to be the same and, thus, their connection strength should be reinforced. In order to extend the original proposals of Hebb rules, Spike-Timing-Dependent Plasticity (STDP) learning rule states that the ordering of spiking events in pre and postsynaptic neurons is a critical aspect when updating the weights. In STDP, the synapse strength is increased if the presynaptic neuron was fired before the postsynaptic cell and, otherwise, decreased if the postsynaptic neuron fired first. In the case of CTRNNs with rate neuron models, the weight updates are computed using the neurons' firing rates or activities (instead of directly using the neuron spikes). A popular Hebbian learning rule, which has been successfully applied in several works (see for instance [94, 95]), is the generalized ABCD Hebbian learning rule.

$$w_{ij}(t+1) = w_{ij}(t) + \eta \, (a_{ij} \, u_i(t) \, u_j(t) + b_{ij} \, u_i(t) + c_{ij} \, u_j(t) + d_{ij}) \tag{2.13}$$

The generalized ABCD Hebbian rule applies locally every weight in the ANN by means of the polynomial transformation of the presynaptic ($u_i(t)$) and postsynaptic ($u_j(t)$) activities exposed in Eq. 2.13. a_{ij} is a parameter that represents the importance of the correlation between neuron activities and b_{ij} and c_{ij} impose how u_i and u_j individually affect the learning. Additionally, d_{ij} is an offset term that defines the weight adaptation under no neuronal activity. Lastly, η is a learning rate that is common to all the synapses and defines the length of the learning steps. To complement Eq. 2.13, Eq. 2.14 expresses the learning rule as a compact vectorial formula. Therefore, using this formulation, the weights of the entire ANN can be updated at once.

$$\mathbf{W}(t+1) = \mathbf{W}(t) + \eta \, \left[\mathbf{A} \odot (\mathbf{u}(t) \, \mathbf{u}(t)^\top) + \mathbf{B} \odot (\mathbf{u}(t) \, \mathbf{1}^\top) + \mathbf{C} \odot (\mathbf{1} \, \mathbf{u}(t)^\top) + \mathbf{D}\right] \tag{2.14}$$

In this case, $\mathbf{u}(t)$ is the vector of firing rates (as in Eq. 2.8) and $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$ and $\mathbf{D}$ are weighted adjacency matrices with the corresponding parameter, a_{ij}, b_{ij}, c_{ij} and d_{ij}, respectively, as weights of the matrix for each connection. Additionally, $\odot$ is the element-wise product and $\mathbf{1} = (1, \ldots, 1)^\top$.

Besides using Hebbian learning rules for learning the suitable weights to address either supervised, unsupervised or reinforcement learning problems, their combination with neuroevolution can drastically enhance the resulting fitness and achieve much more complex solutions. The combination of learning and evolution, apart from being biologically plausible, can smooth and shape the search space and, thereafter, increase the convergence speed of the evolutionary process [96]. There are multiple examples in the literature in which evolution and lifetime learning have been used jointly (see e.g. [94, 95]). In [94], the authors evolved the parameters of the generalized ABCD Hebbian model for both a quadruped locomotion task and a 2D car driving experiment. Moreover, they showed that successful policies can be evolved even with randomly sampled weights. In addition, the authors of [95] extend the NEAT algorithm to the evolution of plastic ANNs. They also used novelty search to mitigate the bootstrap problem and a generalized ABCD Hebbian learning rule, albeit the rule parameters of the connections are not evolved in this case. In our experiments, we use the generalized ABCD Hebbian learning model with both connection weights and learning rule parameters evolved using NEAT algorithm. There is no phenotype to genotype

communication, meaning that the learned weights after the individual lifetime are not transmitted to the evolutionary algorithm and inherited by the individual offspring.

Chapter 3

The Evolutionary Swarm Robotics Simulator

This chapter describes the implementation details, main features and mathematical aspects of the swarm robotics simulator designed and implemented in this Master Thesis. Firstly, the main parts and structure of the simulator are explained. More precisely, the main topics to be treated are: (**1**) The environment where the experiments are carried out, including the objects to be instantiated in the arena. (**2**) The physics engine that allows the realistic physics simulation and collision detection. (**3**) The artificial neural networks explained from an implementation point of view and (**4**) the description of how evolutionary computation techniques are included within the simulator. Lastly, the configuration files used to fully describe and define the experiments are exposed as one of the most important parts of the simulator. The second part of the chapter exposes the sensors, actuators, and the communication system, from a mathematical perspective. Firstly, all the sensors and actuators that are available in the simulator are described in detail. Thereafter, the communication system that the agents can harness to solve cooperative problems is presented, from both the transmission and reception sides.

3.1 Simulator Overview

The swarm robotics simulator presented in this Master Thesis is a highly more complex and versatile version of a previous robotics simulator also developed by us in Python programming language (`https://github.com/r-sendra/SpikeSwarmSim`). The former version was an utterly simplified program in which collisions and realistic physics were not included. Moreover, the previous simulator was visualized only by means of 2D graphics. In this new version, all these limitations have been addressed, constructing a much more realistic swarm robotics simulator. More precisely, aside from general coding improvements and error debugging, collisions, 3D realistic physics and 3D graphics have been implemented using the pybullet library [97]. Furthermore, the algorithms used in this Master Thesis and explained in Chapter 2, which are NEAT and Hebbian learning, have been programmed from zero and are also an important contribution. The main parts of the simulator are the physics engine, the environment and the arena objects, the implementation of ANNs and the evolutionary computation. Among all these items, the physics engine, encompassing collisions, 3D physics and 3D rendering is the only part of the simulator that uses an external python library to simplify its implementation. A more detailed explanation of these simulation modules is provided in the remaining of this section. The code corresponding to the current version of the simulator, with all the improvements and upgrades developed in this Master Thesis, can be consulted at `https://github.com/Robolabo/EvoSwarmSim`.

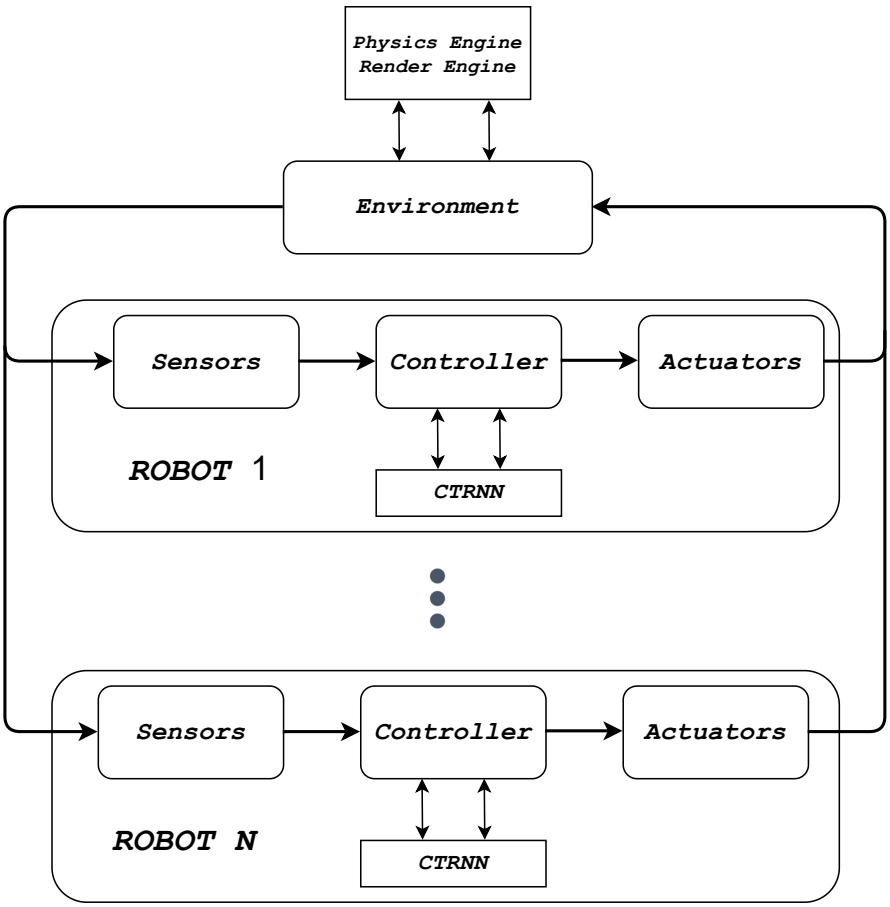


Figure 3.1: General data flow of the simulation.

Moreover, and in order to clarify the overall functioning, Fig. 3.1 shows the data flow of the simulation. In every simulation cycle (corresponding to a loop in the diagram), all the robots in the swarm are updated by means of computing the sensor readings, processing the sensed information throughout the robot controller to generate actions and interacting with the environment using the actuators. In our simulator, the robot controller is normally a neural controller, meaning that the mapping of sensor readings to actions is performed using a neural network (a CTRNN in this case). After the update of every robot, the scheduled actions are managed by the environment, that using the physics engine updates the environment state (positions, orientations, and so on) of every robot.

3.1.1 The environment

The 3D environment where all the experiments are carried out is a square arena of 3m×3m where the robots, lights, and the rest of objects are placed. The arena is limited by 4 walls so that the area where robots can move is constrained. Fig. 3.2 shows an example of the arena with 6 robots, 7 blue cubes, 3 lights of diverse colors and two ground areas. The area where the robots can move is limited by the grey walls (there is a fourth wall below the camera and not shown in the frame). The center of coordinates $(0, 0)^\top$, illustrated in the figure by the reference axis, is the center of the arena.

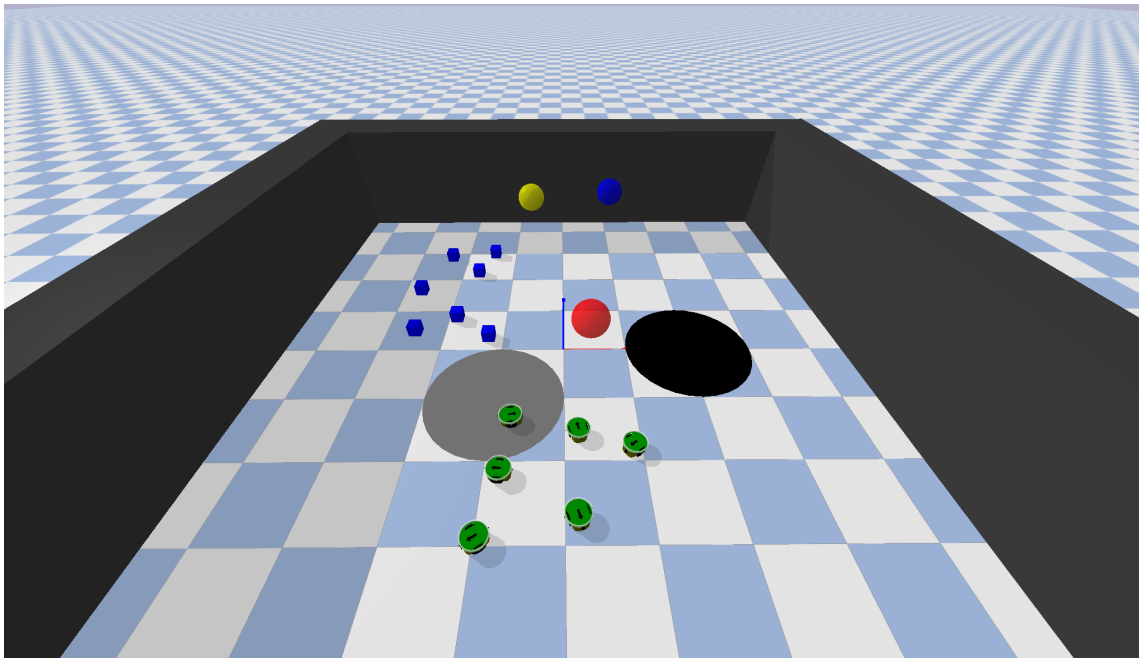


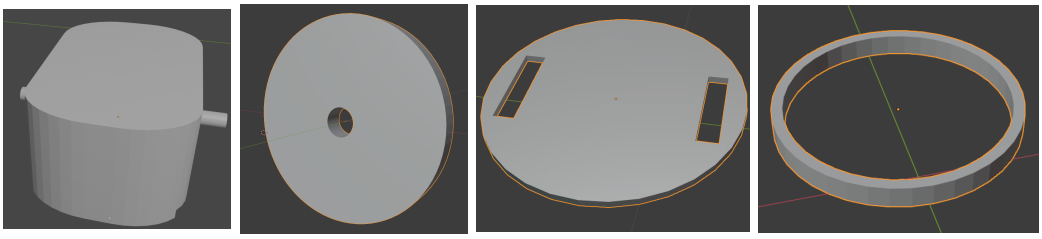
Figure 3.2: Example of an environment with 6 robots, 7 cubes, a red light, a blue light, a yellow light and two ground areas (grey and black).

The most important part of the environment, that allows the design of a wide variety of different experiments and setups, is the set of arena objects belonging to it. The objects are created in the simulation as both python classes designed by us and comprising its non-physics functionalities. Additionally, in order to include the objects into the physics simulation, we use the pybullet function `loadURDF`, that receives a 3D position, an Euler orientation and an URDF file. The Unified Robot Description Format (URDF) files are XML formatted files that describe the links and joints, with their respective geometry, collisions, inertial and visuals, of the objects. Normally, the objects described by URDF files are complex robots, albeit in our simulator we use it to describe any arena object (including robots). The arena objects that are currently implemented in the simulator are the following:

- *Robots*: are the most important objects and the only ones that have sensors to measure the surroundings, a neural controller to generate actions and actuators to materialize those planned actions. Moreover, it is the only object whose parameters can be optimized. The set of robots instantiated in the environment is denoted by $\mathcal{R}$ and every robot is represented by a 3D position $\mathbf{x}_r$ and an orientation θ_r. The term robot is a broad term comprising any controllable object. In the experiments, we use a generic mobile robot as it can be seen in Fig. 3.4a. However, any structure of links and joints can be readily created and included within the simulator. The only requirement is that a new class has to be created inheriting from the class `Robot3D`. Some examples of other robots that can be easily added are a robotic arm, an hexapod or even a simple humanoid. Focusing on the mobile robot, which is the only robot relevant in this Master Thesis, its parts (body, wheels, LED ring, and so on) have been designed and modelled in Blender [1] and combined using the URDF file. The mobile robot has two controllable joints unifying the wheels and the body. An screenshot of the 3D models of the mobile robot pieces can be observed in Fig. 3.3. Even though, under the frame of this Master Thesis, the simulated robot is a generic mobile robot, its modelling takes

[1] `https://www.blender.org/`

inspiration principally from the e-puck robot [2]. In our case, the approximate total mass of the modelled robot, which is important for the physics simulations, is of about 300 g. Viewed from above, the robot has approximately 10 cm of diameter, its height (discounting the LED ring) is about 5 cm and the diameter of the wheels is roughly 5.3 cm.



(a) Body *(b) Wheel* *(c) PCB on top* *(d) LED ring*

Figure 3.3: Screenshots of the 3D modelled parts of the mobile robot in Blender.

- *Light Sources*: are intangible point lights that emit light omnidirectionally. Light sources radiate light of a particular color that can be perceived by the robot light sensors (see Section 3.2) within a range of coverage. The URDF file describing the lights is simply a single spherical link without any kind of collisions and with no mass (so that it can levitate). Visually it looks like a ball of the corresponding color and with some transparency. Notice that the radius of the ball is merely a visual effect to distinguish it in the arena. Fig. 3.4b shows an example of a yellow light source. Provided that the instantiated light sources in the arena are denoted by the set $\mathcal{L}$, then a light source $\ell \in \mathcal{L}$ is represented by a 3D position $\mathbf{x}_\ell$. This notation will be reused along the remaining of the document.

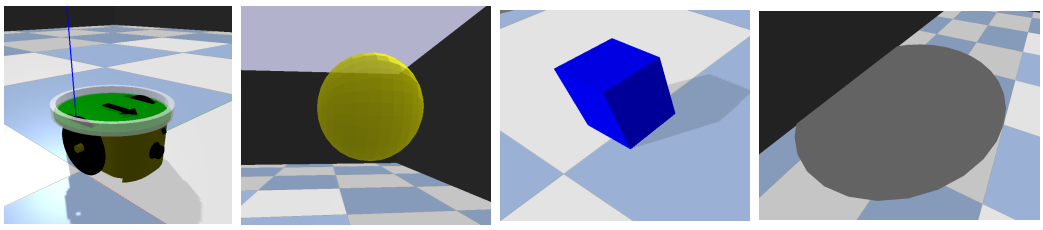
- *Cubes*: are small cubic objects with low mass, so that they can be moved by robots, and colored in a predefined color. We refer to this kind of small objects as cubes because they are used in the experiments, but this item can comprise any object with simple geometries, such as a small sphere or a pyramid. Focusing on the cubes, their mass and side length are configurable and can be varied for each experiment. They are fully colored in blue (or any other color) so that they can be perceived by the color sensor of robots (see Section 3.2). Fig. 3.4c shows an example of a blue cube.

- *Ground Areas*: are intangible arena objects that mark a certain area of the environment with a predefined color. Thereafter, robots can sense the area with a ground sensor. The ground areas can be used to design a wide variety of environments and tasks. In our case, we use them as nests or deposit areas in which robots have to collect cubes. In this version of the simulator, only circular ground areas are implemented (see Fig. 3.4d). A ground area gs is defined by a position in the arena $\mathbf{x}_{gs}$ and a radius ρ_{gs}.

- *Walls*: the walls of the arena are obstacles that, unlike cubes or small objects, the robot cannot move or push because of its high mass. In this Master Thesis, wall objects are only used to constrain the arena space where robots can move and experiments take place.

Within the simulator, the environment is represented by the Python class `World`, which essentially acts as a container of all the arena objects in the simulation. Specifically, the

[2] `http://www.e-puck.org/`



(a) A robot *(b) A light source* *(c) A cube* *(d) A ground area*

Figure 3.4: Screenshots of different objects in a simulation.

main roles of `World` are: (i) create and initialize all the objects (robot, lights, and so on) and the physics engine. By initialize, we refer to assigning the initial position of the objects, initializing the robots' neural networks and reset the sensors and actuators, among other functions. (ii) In every simulation cycle, it calls the method of every arena object responsible of updating their inner state. For instance, it calls the `step` method of every robot in the swarm, which results in the reading of sensors, execution of controllers and generation of actions through actuators. The physics engine step or update method is also executed, so that the physics state of the simulator is iterated. (iii) When a simulation is ended and another one takes place, the `World` class is also responsible for resetting the overall state of the simulator (object positions, robot orientations, CTRNN membrane voltages, and so on) so that the new evaluation can be accomplished cleanly. Another important aspect, from the software perspective, is that arena objects are arranged in groups of objects as a Python `dict` mapping group names to the list of group members (for instance the swarm and the robots belonging to it). This arrangement remarkably eases the management of objects as groups, so that the same processes are applied equally to every group member. For instance, a clear function of this setup is to initialize the objects as a group and avoid overlapping objects. Additionally, it is highly useful when working with multiple swarms (not accomplished in this Master Thesis).

3.1.2 The Physics Engine

In order to implement collisions and realistic physics simulations, the bullet3 [3] library is leveraged. More precisely, instead of directly using the library written in C programming language, we use the Python module pybullet [97] acting as an API. Pybullet (and bullet3) is an open source library that provides efficient and realistic collision detections and rigid body dynamics simulation among other features. Within the simulator, pybullet is included as a physics and render engine to which the environment queries collision detections among arena objects. It also steps the dynamics of the arena objects. All the updates of the physics engine are accomplished at the end of the environment `step`, when all the robot controllers have been executed. This step is carried out using the `stepSimulation` function of pybullet. Moreover, it should be mentioned that in all the experiments, the time step dt of the simulated dynamics is set to 0.02 and the z-axis gravity is fixed to 9.8.

3.1.3 Artificial Neural Networks

In contrast to Sec 2.1, this subsection overviews how the CTRNNs are implemented in the simulator, highlighting and breaking down the main steps of each neural iteration.

[3] `https://github.com/bulletphysics/bullet3`

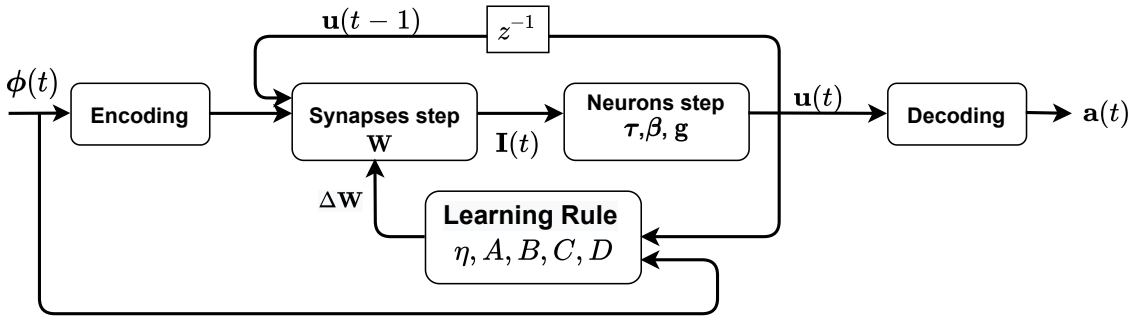


Figure 3.5: Data Flow of the CTRNN with sensor reading encoding, action decoding and synapse learning rule implementation.

Fig. 3.5 illustrates a data flow diagram exposing how artificial neural networks are implemented in the simulation. Firstly, the environment state, composed by the readings of all the sensors, is subject to an encoding stage. The encoding phase encompasses any transformation or preprocessing step that is applied to the state before being fed to the ANN. For example, dimensionality reduction, basis expansions or normalization would lie into this process. It is called encoding instead of preprocessing because its original function was to encode the real valued state $\boldsymbol{\phi}$ into spike trains that can be fed to a spiking neural network. Spiking neural networks are implemented in the simulator and can be used in experiments, albeit we do not explore them in this Master Thesis. Once the state has been encoded, it is fed, jointly with the previous CTRNN state, to the synapses. It can be observed that the synapses step is decoupled from the neuron's update, so that it is only responsible of creating the vector of somatic currents $\mathbf{I}(t)$. In the case of the CTRNNs, the synapses step is simply implemented using numpy as:

$$\mathbf{I}(t) = \mathbf{W}_{\mathcal{N}}\,\mathbf{u}(t-1) + \mathbf{W}_{\mathcal{I}}\,\boldsymbol{\phi}(t)$$

Thereafter, the neurons' state is updated by applying the vector equation exposed in Eq. 2.6, using directly the computed currents $\mathbf{I}(t)$. The result of this step is the new CTRNN firing rates $\mathbf{u}(t)$. This neuronal state is decoded into actions. The most common decoders are the Heaviside decoder, which maps neuron activities into binary actions, or the softmax decoder, which combines the firing rates of multiple neurons to stochastically create categorical actions. The last step of the CTRNN iteration is the update of the synapse weights by means of the Hebbian learning rules. The weight updates $\Delta\mathbf{W}$ of learning rules are computed using the vector formulation in Eq. 2.14.

3.1.4 Evolutionary Algorithms and Parallelization

All the evolutionary computation algorithms implemented in the simulator have three main steps. Besides NEAT, which is used in this Master Thesis, canonical GA, canonical EA [98], Covariance Matrix Adaptation Evolution Strategies (CMA-ES) [99], eXponential Natural Evolution Strategies (xNES) [100] and Separable Natural Evolution Strategies (SNES) [101] are also implemented and can be used in the simulator. The main steps of every algorithm are the following:

- *Genotype to Phenotype conversion*: The first step is to convert the genotypes of the population into artificial neural network phenotypes that can be evaluated afterwards. This mapping is implemented throughout a Python class called `GeneticInterface`, which as its name denotes, acts as an interface between the ANN and the evolutionary population. The genotype to phenotype conversion depends on the precise encoding used by each algorithm. For instance, if the algorithm is a canonical GA with direct

encoding of weights, the mapping is essentially the conversion of the genotype vector into the CTRNN weighted adjacency matrix. In the case of NEAT, whose genes also express topological information, the mapping is more complex. Firstly, the list of genes is mapped into the directed graph of the CTRNN, $\mathcal{G}_{\text{RNN}}$. Thereafter, the adjacency matrix is constructed using the suitable weights expressed in the connection genes.

- *Evaluation*: the genotypes of the population are evaluated in the experiment being addressed. Every evaluation, which may be composed by multiple simulation episodes or trials, leads to a fitness score that states how well the individuals have performed. The evaluation step is implemented as a function (not a class method) called `run_worker`, which is common to all the evolutionary computation algorithms.

- *Population update*: in this step, the population genotypes are subject to the genetic operators explained in Chapter 2. Every evolutionary algorithm implemented in the simulator has its own update method, which results in the new population for the next generation. In the case of the NEAT algorithm, the steps of the update are the selection, the crossover, the mutation and the speciation processes.

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. We use population level parallelization, which 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 parallelization is implemented by means of two alternative frameworks, namely, the multiprocessing Python library [4] and the Message Passing Interface (MPI) Python implementation [5]. More precisely, MPI is used in the experiments of this Master Thesis. In MPI, all the cores, denoted by ranks, execute the program. However, the programmer can specify extracts of code that are only executed by certain cores. Moreover, the execution can depend on the precise rank of the processing unit. Specifically, the population is partitioned so that different minibatches are assigned to each rank. Consequently, every rank executes the same `run_worker` function but with a different genotype. Similarly, the environment object and the physics engine are also different in order to avoid resource collisions.

3.1.5 The Configuration File

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

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

file is mainly divided into `topology`, where all the CTRNN 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 subfield called `populations` in which different evolutionary subpopulations can be specified, allowing the use of cooperative coevolution (see [81]). An example of the `algorithm` field of the JSON field can be observed in the listing below. There, all the hyperparameters of NEAT are fixed and the fitness function is specified.

```
"algorithm" : {
  "name" : "NEAT",
  "evolvable_object" : "robotA",
  "population_size" : 300,
  "generations" : 2000,
  "evaluation_steps" : 4000,
  "num_evaluations" : 8,
  "fitness_function" : "task_switching",
  "populations" : {
    "p1" : {
      "objects":["synapses:weights:all","neurons:bias:all","neurons:tau:all","learning_rule:params:all"],
      "max_vals" : [4,  3, 1.2, 2],
      "min_vals" : [-4, -3, -0.9, -2],
      "params" : {
          "encoding" : "real",
          "p_weight_mut" : 0.1,
          "p_node_mut" : 0.02,
          "p_conn_mut" : 0.15,
          "compatib_thresh" : 0.35,
          "c1" : 1,
          "c2" : 1,
          "c3" : 0.6,
          "species_elites" : 2
      }
    }
  }
}
```

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.
`learning_rule:params:all`	The parameters of the generalized ABCD Hebbian rule of all synapses.
`neurons:decoding:all`	The decoding weights when spiking neuron models are used (not discussed in this Master Thesis).

Table 3.1: Queries to specify the set of parameters to be evolved by a subpopulation in an evolutionary computation algorithm of the simulator.

The subfield `objects` indicates semantically those parts of the CTRNN that are subject to evolution. In the example, those parts are the synapse weights, the neuron biases, the neuron time constants and the learning rule parameters. The subfields `max_vals` and `min_vals` establish the range of the search space. Thereafter, the meaning of the semantics of the `objects` subfield is subsequently explained as a clean and straightforward way of referring to precise parts of the CTRNN. It is based on simple queries of parts of the CTRNN parameters with the semantics exposed in Table 3.1. It gathers all the possible queries that can be performed 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 $\Longrightarrow$ `GET synapses:weights:all`
- Set weights to the values $\Longrightarrow$ `SET synapses:weights:all new_weights`
  in the array variable `new_weights`

This query system is the basis of the phenotype-genotype interface.

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 task_switching.json, resuming a previously saved checkpoint of the evolution and with 20 cores and in verbose mode:

```
$ python main.py -f task_switching --resume --ncpu 20 -v
```

Table 3.2 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	Abbreviation	Arg. Type	Default Value	Description
cfg	−f	String	checkpoint	Name of the JSON configuration file to be used.
ncpu	−n	Integer	1	Number of cores to be used.
resume	−r	Boolean	False	Whether to resume stored checkpoint or not.
render	−R	Boolean	True	Whether to run simulator in visual mode or console mode.
eval	−e	Boolean	False	Whether to run simulator evaluation mode or optimization mode.
verbose	−v	Boolean	False	Whether to run simulator in verbose mode or not.

Table 3.2: Command line arguments of the simulator.

3.2 Sensors

Agents are endowed with a set of sensors that can be used to partially observe the environment. The readings of the sensors are thereafter fed to the neural controller in order to produce

suitable actions. The sensors are generally able to sense only the surroundings of the robots (local perception) and cannot have access to global information about the environment. For instance, a robot is unable to perceive neither its own global position, $\mathbf{x}_r(t)$, within the arena nor its global heading orientation, $\theta_r(t)$. The sensors that are available and programmed in the robotics simulator are the light sensor (LS), distance sensor (DS), ground sensor (GS) and color sensor (CS). In the remaining of this section the functioning of these sensing units will be explained in detail. The communication receiver (RX), which jointly with the communication transmitter endow the robots with direct communication capabilities, is described in Section 3.4. Notice that the use of all the available sensors in the simulator is not a requirement, and robots can leverage only a subset of the sensors depending on the task to be solved. As an example, a task in which there is a single robot that has to find a light source only needs to make use of the light sensor. Even though all the sensor mechanics are elaborated in the remaining of this subsection, Table 3.3 gathers and summarizes a brief description of the available sensors. Let the set $\mathcal{S} = \{LS, DS, GS, CS, RX\}$ denote the set

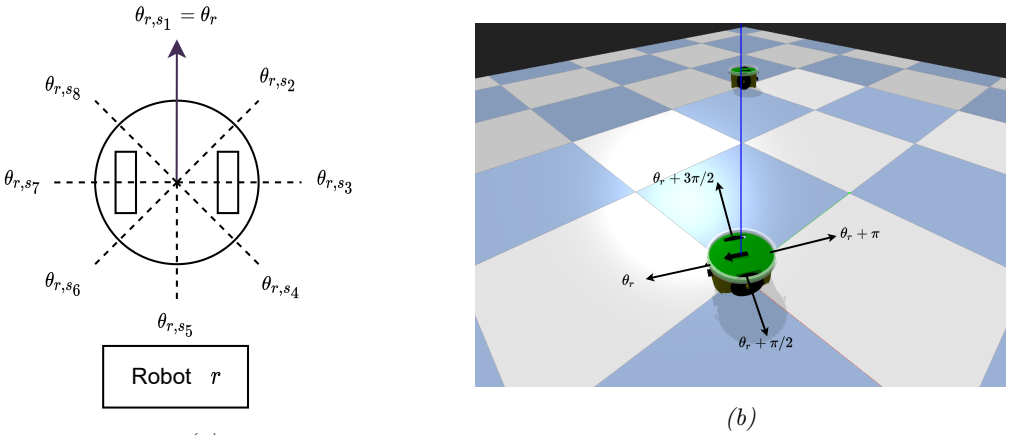
Sensor	Description
LS	Measures the light intensity emitted from light sources. The sensor can be sensitive to different light colors.
DS	Measures the distance to other objects in the surroundings.
GS	Detects whether there is a ground area of a certain color (generally grey or black) underneath the robot.
CS	Detects whether there are objects of a particular color in the surroundings. In our experiments it is used to detect blue cubes in the robot area of perception.

Table 3.3: Description of the available sensors.

of available sensors in the simulator. Moreover, the subset of sensors used in an experiment E is called $\mathcal{S}_E \subseteq \mathcal{S}$. For instance, in the previously mentioned example of the light seeker, the subset would be $\mathcal{S}_E = \{LS\}$. Alternatively, the sensors are sectorized, meaning that the coverage is split into equiareal sectors so that the agent can distinguish the orientation from where a reading is sensed (e.g. the direction with largest light intensity or the orientation of an obstacle perceived by DS). The only sensor that is not sectorized, among those available in the simulator, is the ground sensor (GS). The perceived orientation is discretized and restricted to N possible orientations, where N is the number of sectors. This means that, there is an unavoidable quantization error, which is $\frac{\pi}{N}$ at most, that can be decreased by augmenting the number of sectors. More precisely, denoting $\theta_{r,s_j}(t)$ as the reading of the j-th sector of sensor $s \in \mathcal{S}$ (e.g., $\theta_{r,DS_j}(t)$ or $\theta_{r,LS_j}(t)$), the orientation of the sector is specified by Eq. 3.1.

$$\theta_{r,LS_j}(t) = \theta_r(t) + (j-1)\,\frac{2\pi}{N}, \quad \forall j \in \{1, \ldots, N\} \tag{3.1}$$

These orientations state the angle towards which the sector is most sensitive. Notice that the reading orientation is relative to the heading orientation of the robot ($\theta_r(t)$). In all the experiments and for all the sensors, we fixed the number of sectors N to 4. Thereafter, the possible orientations are $\{\theta_r, \theta_r+\pi/2, \theta_r+\pi, \theta_r+3\pi/2\}$ radians. Fig. 3.6 shows two examples of sectorized sensors as a diagram of the robot, in (a), and in a real simulation, in (b). In the former case, the diagram depicts a robot with 8 sectors and, in the latter scenario, the picture exposes a robot with 4 sectors.



(a) *(b)*

Figure 3.6: Visualization of a robot's sectorized sensor. (a) Diagram showing a zenithal view of a robot with 8 sectors. The dashed lines represent the location of the sensors in each sector. The heading orientation is depicted by the arrow pointing vertically. (b) Example of a simulated robot with 4 sectors. The arrows denote the possible orientation from where the sensor can read the environment. The heading orientation of the robot is indicated by the thick arrow on top of the robot.

Each sensor will result in a measurement tuple $\boldsymbol{\phi}_s(t) \in [0,1]^N, \forall s \in \mathcal{S}$ (for instance, $\boldsymbol{\phi}_{LS}(t)$ in the case of the light sensor) so that its components are the measured values of each sector. Consequently, the overall measurement vector, denoted as $\boldsymbol{\phi}(t)$, is the vector concatenation of the measurements of all the sensors in $\mathcal{S}_E$, as in Eq. 3.2.

$$\boldsymbol{\phi}(t) = \text{vecconcat}\left(\{\boldsymbol{\phi}_s(t),\ \forall s \in \mathcal{S}_E\}\right) \tag{3.2}$$

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

3.2.1 Distance Sensor

The distance sensor (DS) imitates and simulates how IR sensors are used to approximate the distance to objects. The distance estimation is accomplished by emitting an IR beam using an IR LED and computing the time elapsed until an IR detector receives a reflected beam. For each sector j, the reading of DS belonging to robot r is computed as in Eq. 3.3.

$$\boldsymbol{\phi}_{DS,j}(t) = \max_{o \in \mathcal{O}} \left\{ e^{-\lambda^{\rho}_{DS}\, \rho_{ro}(t) - \lambda^{\varphi}_{DS}\, \varphi_{ro,j}(t)^2} \right\} + \varepsilon_{DS} \tag{3.3}$$

Firstly, $\mathcal{O}$ is the set of simulated arena objects with mass. Specifically, in the current state of the simulator, this set encompasses all the robots, all the cube objects and all the walls of the arena. These are the arena objects that can reflect IR beams and therefore be perceived by the DS sensor. In addition, $\rho_{ro}(t) = \|\mathbf{x}_r(t) - \mathbf{x}_o(t)\|_2$ is the Euclidean distance between the sensing robot r position and the targeted object position. $\varphi_{ro,j}(t)$, or misalignment, is the circle distance between the angle of the vector $\mathbf{x}_o(t) - \mathbf{x}_r(t)$ and the orientation of sector j (which is discretized to the set $\{\theta_r,\ \theta_r + \pi/2,\ \theta_r + \pi,\ \theta_r + 3\pi/2\}$). Formally, the computation of $\varphi_{ro,j}(t)$ is governed by the following equation:

$$\varphi_{ro,j}(t) = \min\left\{ |\theta_{r,DS_j}(t) - \measuredangle(\mathbf{x}_o(t) - \mathbf{x}_r(t))|,\ 2\pi - |\theta_{r,DS_j}(t) - \measuredangle(\mathbf{x}_o(t) - \mathbf{x}_r(t))| \right\} \tag{3.4}$$

where $\measuredangle$ is the angle of a vector with respect to the unit vector $(1, 0)^\top$.

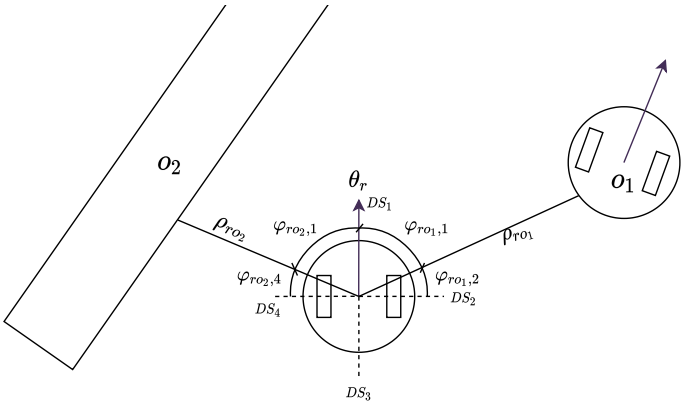


Figure 3.7: Diagram showing the meaning and computation of the misalignment $\varphi_{ro,j}$. The example highlights the misalignment with respect to another robot o_1 and a wall o_2.

Fig. 3.7 illustrates an example in which the meaning of ρ_{ro} and $\varphi_{ro,j}$ are clarified. It shows how these variables are computed in some of the sectors against another robot o_1 and an arena wall o_2. Besides, λ^{ρ}_{DS} and λ^{φ}_{DS} are fixed decaying constants that define and calibrate the coverage and aperture of the sensor. It is important to mention that the $\max\{\cdot\}$ operation in Eq. 3.3 is used to select the nearest object within the sector coverage. Although an additive field would be closer to reality, selecting the nearest object is a sufficiently good tradeoff between a realistic simulation of the reading and computational complexity (notice that computing the measurement to every other object scales poorly as the number of robots increases). Finally, $\varepsilon_{DS} \sim \mathcal{N}(0, \sigma^2_{DS})$ is a normally distributed measurement noise. In the distance sensor, the constants λ^{ρ}_{DS} and λ^{φ}_{DS} are fixed to 0.5 and 1.5, respectively, and the noise standard deviation σ_{DS} is established as 0.1.

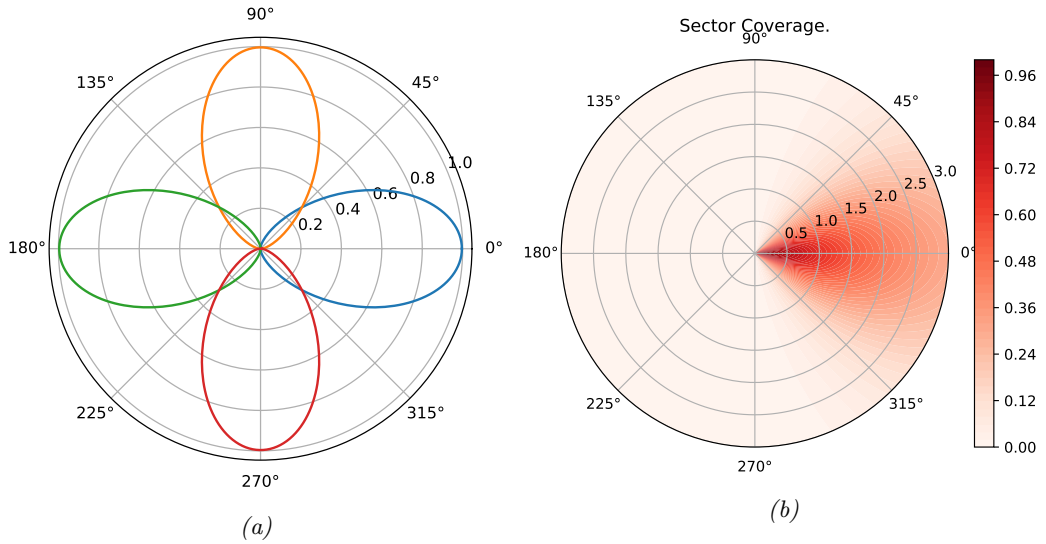
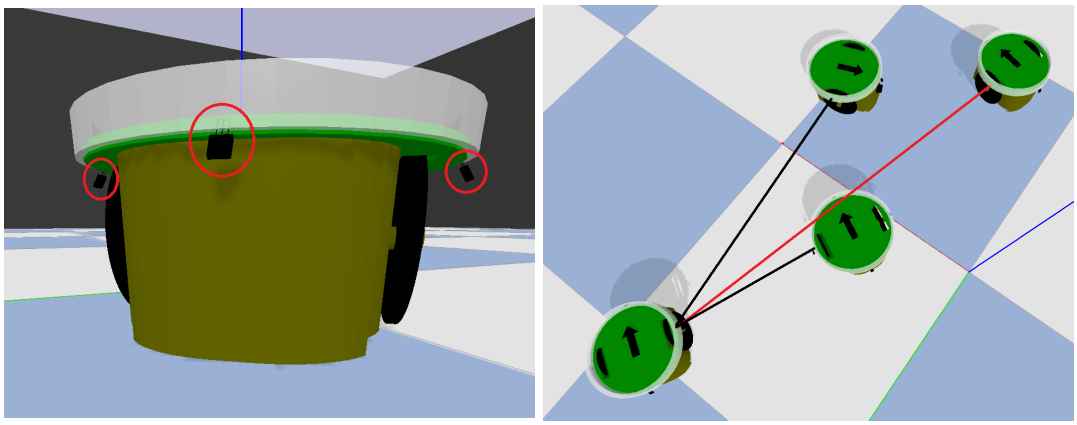


Figure 3.8: (a) Directivity pattern of the distance sensor. Each color represents a sector. (b) Coverage of a single sector of the distance sensor.

In a noiseless scenario, Fig. 3.8 shows the simulated directivity and coverage of the sensor. In Fig. 3.8a, the directivity is shown with a different color for each sector. In Fig. 3.8b, it can be observed how the received signal is attenuated as both distance and misalignment increase. In this case, only the coverage of a sector is presented.

Lastly, an important aspect of the distance sensor functioning is how it detects and manages obstacles between the robot sensing and the targeted object. For instance, in a situation with three robots aligned, those in the extremities should be only able to detect the robot in the middle. In order to implement this feature, and as a novelty from previous versions of the simulator, for each arena object in the range of the sensing agent the `rayTest` function of pybullet is leveraged. `rayTest` casts a ray from two 3D coordinates and verifies the existence of any object intersecting the ray. Therefore, if the function detects an obstacle, the targeted object cannot be detected by the sensor. The robot using the sensor can also be detected itself as an obstacle. For a much more realistic simulation of this feature, we included 3D objects of the small IR devices in each sector of the robots.



(a) *(b)*

Figure 3.9: (a) 3D models of the IR devices used as one of the extremities to cast obstacle detection rays. There is one device in each sector. (b) Example of an obstacle detected with the casted ray from one of the sectors.

Fig. 3.9a highlights the modelled IR devices in each sector of a robot (the fourth one is not shown because it is on the opposite side of the robot). These modelled sensors are used as one of the extremities of the casted ray to verify the obstacles. The other coordinate of `rayTest` is the closest point of the targeted object potentially being detected by the *DS* sensor. The closest point is computed using the function `getClosestPoints` between the links of two pybullet objects. Notice that this function is critical when computing the distance to walls in the arena. Fig. 3.9b depicts a situation in which there is a robot intersecting one of the rays casted by one of the sensors of an agent.

3.2.2 Light Sensor

The light sensor (*LS*) detects the intensity of the light emitted by isotropical light sources. The sensor is also sectorized so that it can perceive also the orientation from where the light was measured, among 4 possible angles. Moreover, owing to the fact that there can be light sources of different colors, the light sensors can be also sensitive to a precise color. As in

Eq. 3.3, the readings of LS sensor are modelled using Eq. (3.5) for each sector $j \in \{1, \ldots, 4\}$

$$\phi_{LS,j}(t) = \sum_{\ell \in \mathcal{L}} c_\ell \left(e^{-\lambda^{\rho}_{LS}\, \rho_{r\ell}(t) - \lambda^{\varphi}_{LS}\, \varphi_{r\ell,j}(t)^2} + \varepsilon_{LS} \right) \tag{3.5}$$

where $\mathcal{L}$ is the set comprising all the lights in the arena and $\rho_{r\ell}(t) = \|\mathbf{x}_r(t) - \mathbf{x}_\ell(t)\|_2$ is the Euclidean distance between the position of the sensing robot r and the position of the targeted light source ℓ.

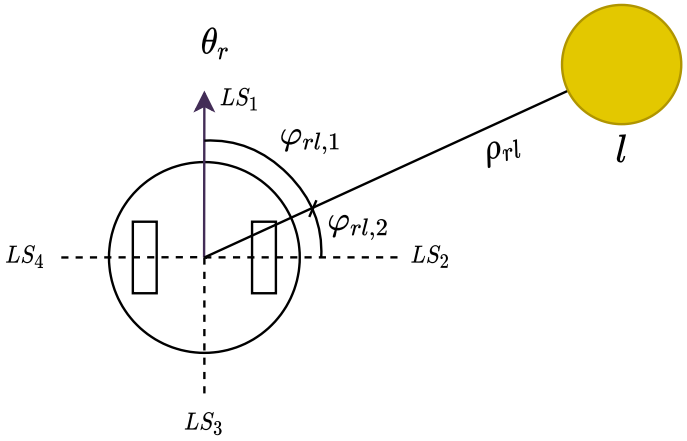
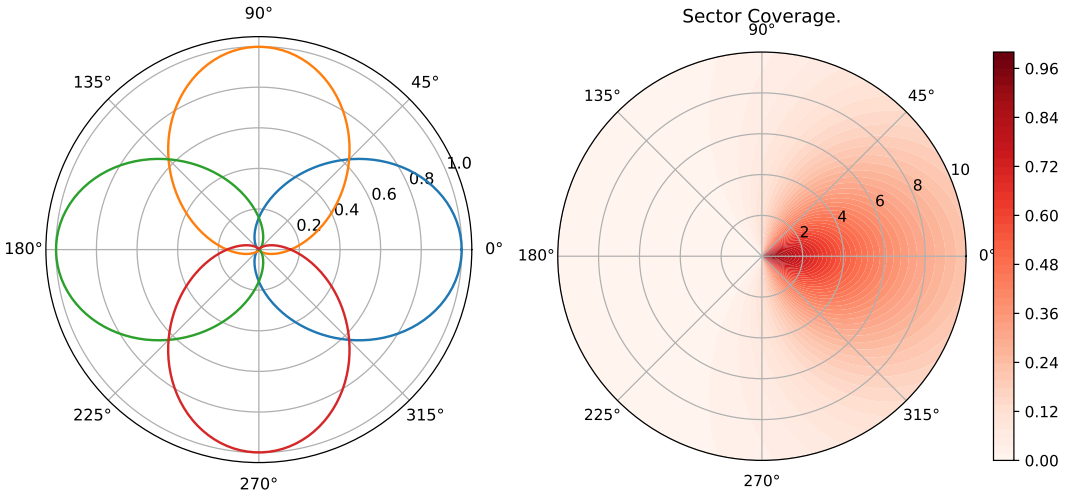


Figure 3.10: Diagram showing the meaning and computation of the misalignment $\varphi_{r\ell,j}$. The example highlights the misalignment with respect to a yellow light source ℓ.



(a) *(b)*

Figure 3.11: (a) Directivity pattern of the light sensor. Each color represents a sector. (b) Coverage of a single sector of the light sensor.

Additionally, $\varphi_{r\ell,j}(t)$ is computed as in Eq. 3.4, albeit instead of using the position of a tangible object ($\mathbf{x}_o$), the coordinates of the light ℓ, $\mathbf{x}_\ell$, are used. Fig. 3.10 exemplifies the computation of $\varphi_{r\ell,1}(t)$ and $\varphi_{r\ell,2}(t)$ in a diagram with a single light source. λ^{ρ}_{LS} and λ^{φ}_{LS} are also decaying constants of each term. In the case of LS, $\lambda^{\rho}_{LS} = 0.15$ and $\lambda^{\varphi}_{LS} = 0.75$. Notice

that compared to the profile of the distance sensor, the chosen parameters in the light sensor allow a wider coverage and a higher sector aperture. Furthermore, the variable $c_\ell \in \{0, 1\}$ is a bit stating if the received light is of the same color as the one towards which LS is sensitive ($c_\ell = 1$). Additionally, it can be observed that the sensed light from all the lights are combined as a summation. Although it is not represented in the equation, the reading of each sector j, $\phi_{LS,j}$ is saturated to a maximum value of 1, so that the measurement is constrained to the interval [0, 1]. Lastly, $\varepsilon_{LS} \sim \mathcal{N}(0, \sigma^2_{LS})$, with $\sigma_{LS} = 0.1$, is the white noise attached to the measurement, where $\mathcal{N}$ denotes the normal distribution. Fig. 3.11 illustrates the directivity of the LS sensor (in (a)) and its coverage (in (b)).

3.2.3 Color Sensor

The color sensor is a high level sensor capable of locally detecting objects of a specific color. In the experiments carried out in this Master Thesis, the color sensor is used mainly to perceive blue cubes distributed in the arena where robots are placed. This sensor is a simplified version of the combination of a camera followed by a simple processing of the captured frames. More precisely, owing to the fact that the required environment information is merely the presence of objects of a particular color, the frame processing would be to detect the amount of blue pixels in the frame. Thereafter, it would detect a blue object in the surroundings if the number of blue pixels surpasses a fixed threshold. Moreover, the number of pixels of the corresponding color can also provide information about the distance to the perceived object. Few blue pixels would probably indicate that the object colored in blue is distant. Alternatively, a frame with a high percentage of blue pixels can represent a scenario with a blue object just in front of the camera. The use of a camera and the mentioned processing is not implemented in our simulation in order to avoid a huge overload and increase of computation time in the system.

The color sensor is also a sectorized sensor with 4 independent sectors, so that the robot can perceive also the approximate direction from where the blue object was sensed. In a real world scenario, the directional sensing can be achieved using a 360 degrees camera. The reading of each sector is binary and returns 1 if there is a blue cube at a distance lower than a threshold $\rho_{c,max}$ = 1m and within the sector area. Otherwise, it reads a 0 if there is no targeted object within the sector. Thus, the reading of this sensor at each time instant t is $\phi_{CS}(t) \in \{0, 1\}^4$.

3.2.4 Ground Sensor

The ground sensor (GS) is a sensor that detects whether or not there is a ground area underneath the robot's body. It can also filter out and perceive only the ground areas of a precise color (normally grey or black). More formally, provided that ground areas are always circles defined by its center coordinates ($\mathbf{x}_{gs}$), a radius (ρ_{gs}) and a color, the GS sensor reads a 1 if it is within the ball centered at $\mathbf{x}_{gs}$ and of a radius ρ_{gs}.

$$\phi_{GS}(t) = c_{gs}\, \mathcal{H}\left(\rho_{gs} - \|\mathbf{x}_{gs} - \mathbf{x}_r\|_2\right) \tag{3.6}$$

Eq. 3.6 represents this computation, where gs is the nearest ground area (robots cannot be in two ground areas simultaneously), $\mathcal{H}$ is the Heaviside function and $c_{gs} \in \{0, 1\}$ states if the sensor is sensitive to the color of gs.

3.3 Actuators

In contrast to sensors, actuators receive the actions planned and established by the robot controllers and materialize them in order to interact with the environment. The actuators

that are currently implemented in the simulator are the joint actuator (JA), for controlling either by velocity or position any joint of the robot, the LED actuator (LD) for turning on or off the robot LEDs and the grasp and drop actuator (GDA) which is a high level actuator for easing the object transportation. Furthermore, the communication transmitter (TX), that can be also considered as an actuator, is presented separately in Sec. 3.4. The set of actuators available in the simulator is denoted as $\mathcal{A} = \{JA, LA, GDA, TX\}$. Moreover, $\mathcal{A}_E$ is the set of actuators that are used in an experiment E. The action generated by the controller and being fed to the corresponding actuator is identified, at time instant t, as $\mathbf{a}_n(t)$ where $n \in \mathcal{A}_E$ is the actuator. For instance, in the case of the joint actuator with, say, 5 joints, the action vector is $\mathbf{a}_{JA}(t) = (a_{j1}, \ldots, a_{j5})^\top$. Each vector component a_{jk} would be the scalar action controlling the robot joint $k \in \{1, \ldots, 5\}$. The overall action vector resulting from the vector concatenation (denoted as vecconcat) of all enabled actuators in an experiment is shown in Equality 3.7.

$$\mathbf{a}(t) = \text{vecconcat}\left(\{\mathbf{a}_n(t),\ \forall n \in \mathcal{A}_E\}\right) \tag{3.7}$$

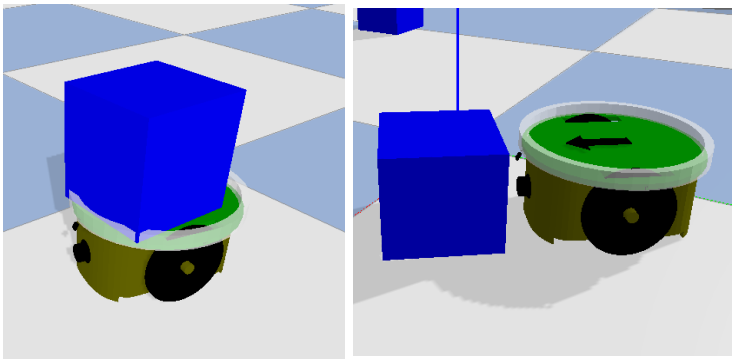
3.3.1 Joint Actuator

The joint actuator (JA) is the actuator responsible for moving the joints of the robot according to the actions generated by the controller. The actuator, which manages all the joints, receives the joint action vector $\mathbf{a}_{JA}(t) = (a_{j1}, \ldots, a_{jK})^\top$ with each scalar action controlling a different joint and where K is the number of joints to be controlled. The control can be either a velocity control or a position control. In the former, a target velocity is established (which is, in turn, the scalar action) and the control system tries to reduce the error between the target velocity and the current rotation speed of the robot's joint. On the contrary, a position control means that the action is a reference angle and the control system aims at reducing the error between a reference angular position and the current joint orientation. Both joint controls are implemented by means of the pybullet function `setJointMotorControl2`. This function takes as arguments the robot id, the joint index and the control mode, which can be either `VELOCITY_CONTROL` or `POSITION_CONTROL`. It also receives the target velocity if the control is by velocity or the target position if it is a position control. In our experiments, we use the velocity control to manage two joints controlling the wheels' rotation of mobile robots. Therefore, the joint actuator simplifies to $\mathbf{a}_{JA}(t) = (a_{j1}, a_{j2})^\top$. Owing to the fact that these actions are generated by a neural controller and are bounded in [0, 1], the target velocity fed to the velocity control is $\boldsymbol{\omega}(t) = \omega_{max}\, \mathbf{a}_{JA}(t)$. ω_{max}, in rad/s is the maximum velocity achievable. This maximum angular velocity is fixed in the experiments to 2rad/s. The velocity control of the two joints of the mobile robot results in movements described by the kinematics of differential drive systems (see e.g. [72]).

3.3.2 Grasp and Drop Actuator

To ease the experiments, the Grasp and Drop Actuator is a high level actuator that utterly simplifies the action of grasping or dropping objects in the environment. More precisely, when a small arena object (for instance, a cube) is very close to the robot, it can be immediately picked or grasped by the robot. Similarly, if an agent has already grasped an object, it can decide to drop it again to the arena. This actuator is an abstraction of a real actuator in which the robot grasps objects using, for example, a simple robotic grasper. The main motivation of this actuator is to allow an straightforward manner of transporting objects across the environment without drastically increasing the complexity of the actuator (which would highly hinder the important topics of this Master Thesis). Clearly, the robot can also move objects by pushing them throughout its joint actuators. Returning to the description of the actuator functioning, only one object can be grasped simultaneously and, at the software

level, each object can only be picked by a single robot in order to avoid resource collisions. Visually, when a agent grasps an object, the object is placed on top of the robot's body. On the contrary, when the object is dropped, it is placed on the arena floor in front of the robot. Very importantly, we decided to include a grasp cooldown, meaning that a robot cannot grasp anything if it has recently dropped an object. Specifically, this cooldown is applied over a time window of 50 simulation cycles. Besides, the robot controller can query three different commands to the grasp and drop actuator. These commands are skip, which means that the actuator will not do any operation; grasp, to pick an object in the robot surroundings or drop, to release an already grasped object. The action representing these commands is a scalar action $a_{GDA} \in \{0, 1, 2\}$, so that 0 means skip, 1 means grasp and 2 means drop. In the experiments, this action is generated by means of applying a softmax operation to a neuronal ensemble with 3 neurons, each one representing a value of the action. Therefore, using the softmax function also includes some randomness into the actuator. Fig. 3.12 shows two screenshots of a robot grasping and dropping a cube, respectively.



(a) Cube grasped *(b) Cube dropped*

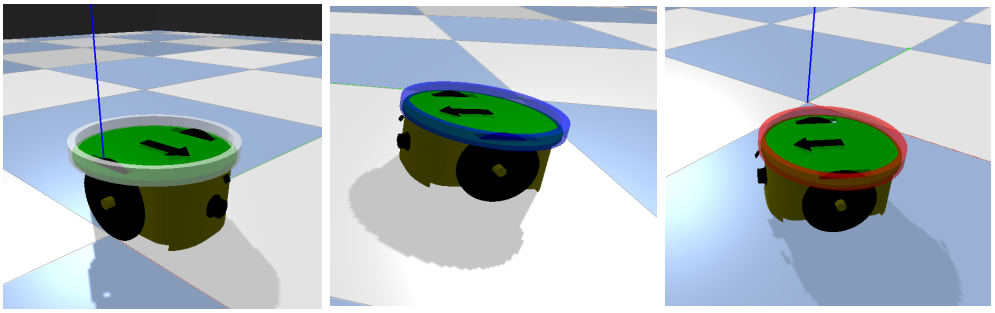
Figure 3.12: Screenshots showing the functioning of the grasp and drop actuator.

3.3.3 LED Actuator

The LED actuator (LA), is controlled with the action $a_{LA}(t) \in \{0, \ldots, C\}$, where C is the number of colors. In this situation, $a_{LA}(t) = 0$ turns off the LED and non zero values of $a_{LA}(t)$ turn on the LED photodiode with one of the C possible colors. If only one color is considered ($C = 1$), then $a_{LA}(t) \in \{0, 1\}$. LED actuators do not aim at serving as communication means. Their objective is to notify to the researcher and to the optimization algorithms the individual solution of the agent against the collective task to be solved. In other words, an agent cannot perceive whether or not there is a neighboring robot activating its LED. The LED is visually represented as a ring on top of the robot's body, so that the user can be aware of the LED action being executed by every robot if the simulation is run in visual mode. The LED ring has an opacity of 0.6 and changes its color according to the LED action a_{LA}. Fig. 3.13 shows simulation screenshots of a robot with the LED deactivated, in (a), with the blue LED activated, in (b), and with the red turned on, in (c).

3.4 Communication System

This section describes the simulated communication system that the robots can harness in order to solve cooperative multi-agent problems. Agents can omnidirectionally emit an M



(a) Deactivated LED *(b) Blue LED* *(c) Red LED*

Figure 3.13: Example of the functioning of the LED ring of the mobile robots.

dimensional message that can be received by other robots in a reduced 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 technologies for both transmission and reception (see e.g. [58]). All the swarm members have, in addition to all the sensors and actuators previously described, an IR communication transmitter and an IR communication receiver. It should be also mentioned that the communication mechanics and the way in which robots leverage the communication system is the result of the evolution process. In other words, we design how robots can communicate and the limits of communication, but the communication semantics should emerge from the optimization process. The communication system proposed in this Master Thesis, inspired by how mobile robots communicate using IR technology, is minimal because of the following main reasons:

- The communication is local. This means that two robots can communicate if their distance is lower than a threshold.
- Only one message can be received at each time instant. 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 main parts of the communication systems, namely, the transmitter and the receiver, are explained in the following.

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 $\mathbf{a}_{TX}(t) \in [0, 1]^M$ be the communication action directly generated by the robot neural controller in response to input sensed stimuli. $\mathbf{a}_{TX}(t)$ represents the elaborated message of length M. In the experiment proposed in this work, the message dimension is set to $M = 1$.

Reception: Agents are able to emit an omnidirectional message $\mathbf{m}^r_{TX}(t) \in [0, 1]^M$ (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 $N_{RX} = 4$ equiareal sectors, so that each of them will perceive only the message of the nearest neighbor within the corresponding sector coverage. Moreover, as explained in the distance

sensor description in Sec. 3.2, we use the pybullet function `rayTest` to compute obstacles between a sender and a receiver. 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 agents 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. 3.3). 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 mechanism is cyclic, meaning that the first message is chosen from sector 1, and thereafter, the selection obeys the following sequence $\{\mathbf{m}_{RX,1}, \mathbf{m}_{RX,2}, \mathbf{m}_{RX,3}, \mathbf{m}_{RX,4}\}$. Thus, after picking the message $\mathbf{m}_{RX,4}$ corresponding to sector 4, $\mathbf{m}_{RX,1}$ is selected again and the sequence is restarted. The message of a sector is also selected in the case that there is no robot in the area of that sector. In such a situation, the message would be zero. According to these statements, the message of every sector is selected every 4 simulation cycles. Owing to the fact that the neural controller receives information from all the sectors at the same time, the following technique is applied: the message fed to the CTRNN from the sectors that are not currently selected is fixed to the last value sensed in that orientation. In other words, the message from a sector is kept constant until the sector is selected for reception again and the message is updated. This process is similar to sampling a signal with a sampling period of 4 simulation cycles.

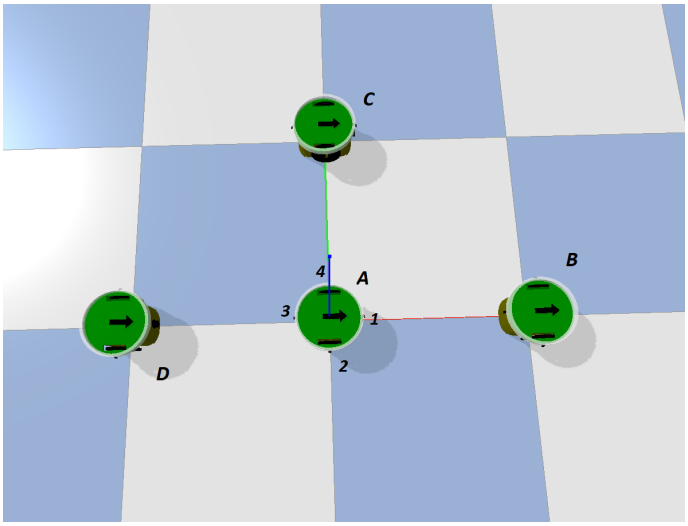
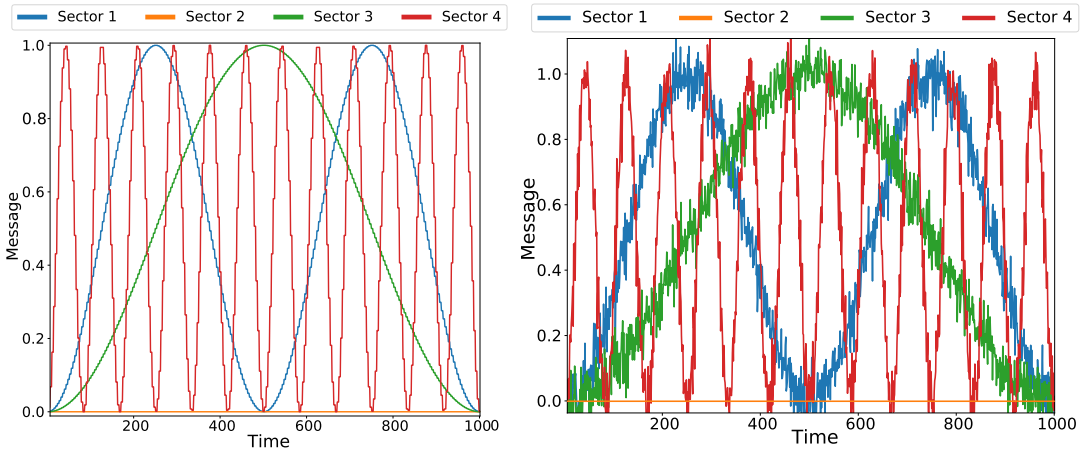


Figure 3.14: Toy example to depict the functioning of the communication. There are three robots and the focus of the example lies on robot A. The numbering of the sectors of robot A is shown, clearly displaying that it has one neighbor all the sectors excluding sector 2.

In order to clarify the communication receiver, we propose the following toy example. The example setup is depicted in Fig. 3.14. showing 4 static robots pointing to the same direction. The robots are labelled as A, B, C and D, as it can be observed in the figure. The focus of the example lies on the robot in the center (robot A), which has neighbors in three of its four sectors. Specifically, as illustrated in Fig. 3.14, the neighbors of robot A are robot

B in sector 1, robot D in sector 3 and robot C in sector 4. Notice that sector 2 of robot A is the area without any other agent.



(a) Received messages without noise.

(b) Received messages with noise.

Figure 3.15: Messages received by robot A of Fig. 3.14 from its four sectors. The messages are illustrated without and with attached white noise.

Fig. 3.15 shows the received communication messages fed to the neural network of the robot in the middle. Firstly, Fig. 3.15a depicts the messages received by robot A without attached noise. The messages are manually selected as sinusoidal signals at different frequencies for the sake of clarity in this toy example. Each robot B, C or D sends a sine wave of a given and unique frequency. Notice that in the actual experiments, the messages are not necessarily sinusoidal because they are generated by the robot's CTRNN. In the figure, it can be noticed that the blue signal is emitted by robot B, the green signal by robot D and the red signal by robot C. The orange signal is always zero because there are no neighboring robots in sector 2 of robot A. In Fig. 3.15a, it can be clearly observed the effect of sampling the received signals. Because the sampling period is very low, the effect of sampling is not extremely lossy unless the messages change remarkably quickly. Fig. 3.15b shows the same example with white noise attached to the messages. Notice that, if the frequency of the message signals is not very high, the noise masks the sampling effect, and the periods where the signal is maintained constant are no longer distinguishable.

Chapter 4

Experiments

This chapter establishes and explains the different experiments that have been accomplished in this Master Thesis using the simulator and techniques of Chapter 3 and the algorithms and models of Chapter 2. It starts by defining the tasks that the robots must solve within each experiment and the experiment mechanics. Subsequently, the fitness functions used to evaluate the performance of the genotypes are exposed and described in detail. Besides, the experimental setup, comprising the neural controller architectures and the hyperparameters of the evolution, are specified.

4.1 Description of the experiments

In the experiments proposed in this section, we address two common and challenging topics in swarm robotics. These topics are (**1**) multitasking and task switching and (**2**) emergence of communication and signalling. In task switching, the evolved robots must be capable of accomplishing multiple independent tasks and being able to switch among them at the right timing. Tasks are solved sequentially and, desirably, all agents have to solve only the corresponding task at each time instant. Furthermore, successfully addressing a task at the wrong timing is not rewarded. All the tasks must be faced by the robots in every simulation and the order of the tasks is randomly selected. Secondly, as it will be explained hereafter, the communication is studied also under the frame of task switching, so that only one robot actually knows the task that has to be accomplished at each simulation cycle. Thereafter, in order to achieve a sufficiently good performance, some communication mechanics must emerge. Two different experiments are proposed and addressed to deal with these two problems (task switching and communication). The first experiment is fully focussed on the task switching aspect. Compared to the second experiment, the tasks of the Experiment 1 are more complex and challenging. On the contrary, the second experiment is mainly devoted to the study of communication emergence. Owing to the fact that the communication introduces an utterly complex dimension to the experiment, highly hindering the evolution process, the tasks in this second experiment are slightly easier and less challenging.

4.1.1 Experiment 1

As previously stated, Experiment 1 addresses the topic of task switching and multitasking. The pool of tasks of the experiment is composed by two tasks, which will be denoted as Task A1 and Task B1. In each simulation trial, the genotype evaluation is split into two time windows of the same length. Provided that the entire simulation lasts T_{E_1} cycles, the first time window is fixed between time instant 0 and $T_{E_1}/2$. The second time window is $[T_{E_1}/2, T_{E_1}]$. In each of these time intervals, the task to be accomplished is different. For

instance, in a potential scenario the robots have to solve task A1 during $[0, T_{E_1}/2]$ and task B1 during $[T_{E_1}/2, T_{E_1}]$. Thus, it is at the simulation cycle $T_{E_1}/2$ when the task switching occurs and the robots' behavior must change. At each simulation, the task that lies in each time window is selected randomly and with equal probability. Specifically, the random selection is accomplished without replacement. This means that, as there are only two tasks and two time slots, the randomness is presented only in the selection of the first task (the one corresponding to $[0, T_{E_1}/2]$). The task of the second time slot is directly and deterministically the one that was not selected in the first interval. The tasks of Experiment 1, to be solved sequentially in each time slot, are described in the following:

- **Task A1**: rewards robots if they closely follow a mobile red light source. The light source, as described in Section 3.1, emits red light omnidirectionally and can be sensed by the robots' red light sensor. The movement of the light source is a simple orbit around the central coordinates (0, 0) of the arena. The kinematics of the light are, therefore, summarized in Eqs. 4.1 and 4.2. We express the position update in polar coordinates ϑ and r dependent on the simulation time instant t:

$$\left.\begin{aligned} \vartheta(t+1) &= \vartheta(t) + \Delta\vartheta \\ \mathsf{r}(t+1) &= \mathsf{r}(t) + \frac{(R - \mathsf{r}(t))}{50} \end{aligned}\right\} \tag{4.1}$$

  where, considering (0, 0) as the origin of coordinates, $\mathsf{r}(t)$ is the instantaneous radius of the trajectory and $\vartheta(t)$ is the angle in radians. $\Delta\vartheta = 0.01$ is the angle increment at each iteration and $R = 1.5$m is the target radius towards which the light trajectory tends. This means that the radius will grow exponentially until R is reached. Thereafter, it will remain steady at $\mathsf{r}(\infty) \approx R$. Provided that $\mathsf{r}(0) = 0$, the trajectory of the light source is the one illustrated in Fig. 4.1.

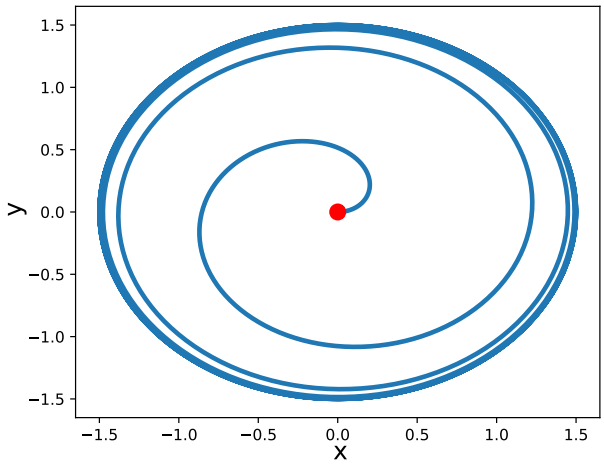


Figure 4.1: Trajectory described by the red light source of Task A1 of Experiment 1. The red dot indicates the starting position.

The previous polar coordinates are converted into a cartesian position of the light as in Eq. 4.2.

$$\mathbf{x}_\ell(t) = \mathsf{r}(t) \begin{pmatrix} \cos(\vartheta(t)) \\ \sin(\vartheta(t)) \end{pmatrix} \tag{4.2}$$

The initial values of the dynamics are always fixed to,

$$\vartheta(0) = 0, \mathsf{r}(0) = 0 \Rightarrow \mathbf{x}_\ell(0) = (0,\, 0)^\top$$

- **Task B1**: is a transportation task in which robots have to collect small blue cubes and transport them to a common nest area. The number of agents never exceeds the amount of cubes and each robot can transport each cube by its own, without any required aid or physical contribution from other swarm members. The nest area, where the solid cubes should be gathered, is a circular grey ground area with a yellow point light source above. The light source has a coverage range large enough so that robots can sense it from any point in the arena. This light acts as a beacon, notifying the agents the location of the nest. Both the nest ground area and the yellow light source are clearly static. In order to slightly hinder task B1, three different grey ground areas are placed in the arena at fixed positions, while there is only one yellow light source randomly situated above one of them. Consequently, at each simulation trial, the correct nest area, where cubes have to be transported, is the ground area underneath the yellow light source. Placing the objects in the incorrect area will not be neither penalized nor rewarded. Finally, as explained in Section 3.2, the blue cubes can be perceived by the robots by means of the binary reading of the color sensor. As this sensor unit is sectorized, the agents can also acquire the orientation where the cube is (among 4 possible angles). Moreover, agents can transport cubes either by pushing them or harnessing the grasp and drop high level actuator presented in Section 3.3.

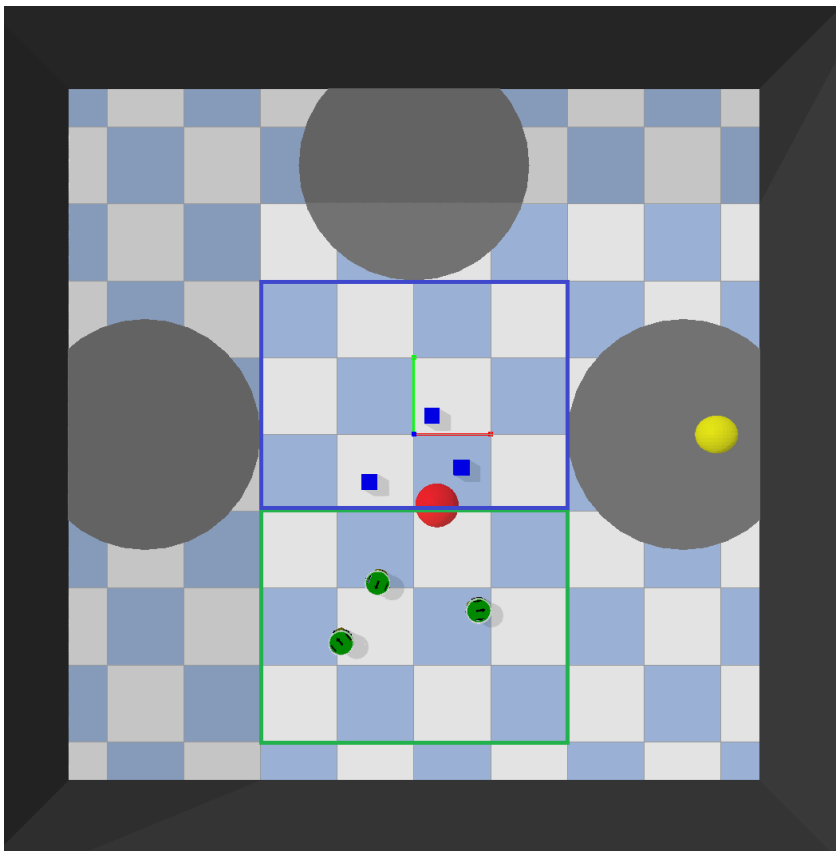


Figure 4.2: First frame of the Experiment 1 arena with a zenithal view.

An important remark is that robots are aware of the current task to be solved by means of a binary input to the CTRNN, in which a value of 0 represents task A1 and 1 denotes task B1. Therefore, one of the two possible scenarios would be that all the robots start seeking and following the red light source until time step $T_{E_1}/2$. Thereafter, the swarm members drastically should modify their behavior by ignoring the red light and trying to transport the cubes to the nest. The other possible situation would be starting by collecting the cubes and finishing with the red light pursuit. Fig. 4.2 shows a screenshot of the first frame of a simulation of Experiment 1. It can be appreciated that the red light source, depicted as the red ball, starts at the center of the arena. Moreover, there are 3 ground areas at the north, west and east of the environment, but only the one on the right side has a yellow light source (yellow ball) above. Thus, robots have to transport the blue cubes in the middle of the arena to the eastern ground area. Due to the zenithal perspective, the robots are shown as green circles with a black arrow denoting their heading orientation (see Fig. 3.4 for a clearer view of the robot appearance). Lastly, an important aspect to be highlighted is how the arena objects are initialized within the environment. The grey ground areas are always positioned at the coordinates shown in Fig. 4.2. Moreover, the red light deterministically starts at the center position (0, 0). On the contrary, the yellow light is stochastically positioned at the center of one of the ground areas, with equal probability. This means that the agents have to transport the cubes to a potentially different place in every simulation. The positions of the cubes and the robots are uniformly initialized in the squares highlighted in Fig. 4.2 (the blue area for the cubes and the green area for the robots). The only constrain is that, in the random sampling, two objects (robots or cubes) cannot be initialized at the same position or at any pair of coordinates that lead to collisions and physical overlapping. The orientation of the agents is also random and sampled from $\mathcal{U}(0, 2\pi)$. Lastly, the experiment duration T_{E_1} is fixed to 2000 time steps in total. This means that each time window, in which the tasks are solved, lasts 1000 simulation cycles.

4.1.2 Experiment 2

The second experiment also addresses the problem of task switching with 2 tasks. The simulation duration T_{E_2} is partitioned again into two time slots $[0, T_{E_2}/2]$ and $[T_{E_2}/2, T_{E_2}]$ and a different task has to be accomplished within each time window. However, the most significant difference with respect to Experiment 1 is that multitasking is combined with the study of communication among robots. More precisely, only one of the robots knows the task that has to be addressed in each time slot and, consequently, it has to communicate this information to the rest of the swarm. The communication is implemented by means of the communication system exposed in Sec. 3.4. This communication system defines the rules and mechanics of the direct interaction, albeit the way in which robots use it to cooperate has to emerge from evolution. As in the previous experiment, the current task is notified to the CTRNN of the robots as an input with binary encoding. Nonetheless, in this scenario, those robots that do not know which task to execute receive always a value of -1. The addition of the communication involves a remarkable rise of the complexity of the problem. Therefore, as a first approach to the topic, in this Master Thesis some slight simplifications were required. Thus, in order to ease the experiment, the two tasks to be solved are both the pursuit of a light source as in Task A1 of Experiment 1. However, in order to create two differentiated tasks, the color of the light to be followed is different depending on the time slot. Firstly, in Task A2 the swarm has to search and pursuit a red light source, while in Task B2 the color of the light source to be followed is yellow. Fig. 4.3 depicts the first frame of a simulation of this experiment, where the two lights and the robots can be observed.

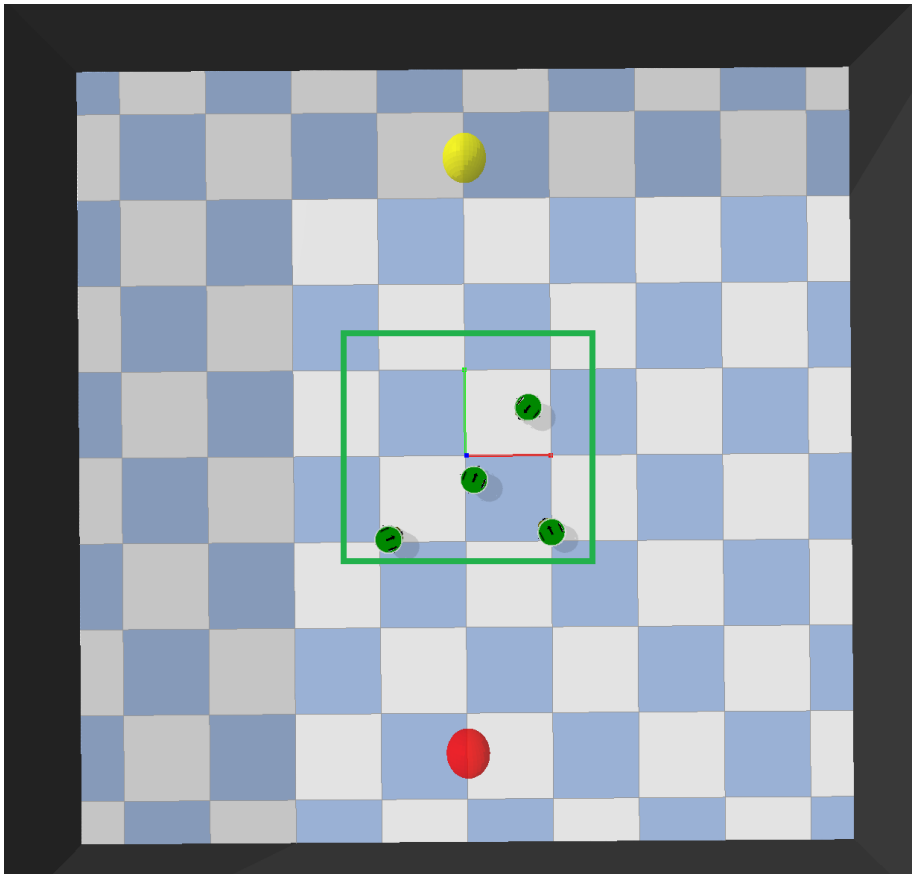


Figure 4.3: First frame of the Experiment 2 arena with a zenithal view.

In this second experiment, the lights are initialized deterministically at the positions exposed in Fig. 4.3. Both lights describe a clockwise circle orbit around the center of the arena. In order to maximize the distance between lights, they are initially placed at counterphase. Their orbit essentially obeys the trajectory described in Eqs. 4.1 and 4.2, albeit as they are already initialized at the steady state radius $R = 1.5$, the radius dynamics can be omitted. Lastly, robots' positions are randomly sampled from an uniform distribution in the area highlighted in green in Fig. 4.3. Robot heading orientations are also initialized stochastically. The duration of the simulations of Experiment 2 is also defined as $T_{E_2} = 2000$.

4.2 Fitness Functions

This section presents and explains the fitness functions used to evaluate genotypes in the previously defined experiments. The fitness functions to be determined are the light following and the cube transportation fitness functions. The former corresponds to tasks A1, A2 and B2 while the latter is applied to task B1.

- **Light pursuit fitness function**: Firstly, two terms are considered in the fitness function for the light pursuit task. These terms are: (**1**) the fitness evaluator for the steady state behaviors, when robots should have had enough time to find and move towards the light position and (**2**) a transit fitness function that more loosely evaluates the robots at the initial time steps of the task, when there is a considerable distance between them and the light source. Eq. 4.3 states the overall fitness function of the

light pursuit task, comprising both steady state and transit terms.

$$F_1 = \frac{1}{100\,R}\sum_{t=0}^{99}\sum_{r\in\mathcal{R}}\max\left\{1-\frac{\|\mathbf{x}_\ell(t)-\mathbf{x}_r(t))\|_2}{\rho_l(t)},\,0\right\} \tag{4.3}$$

$$+\,\frac{1}{(T_E/2-100)\,R}\sum_{t=100}^{T_E/2}\sum_{r\in\mathcal{R}}\mathcal{H}\left(\rho_\ell(\infty)-\|\mathbf{x}_r(t)-\mathbf{x}_\ell(t)\|_2\right)$$

The first line of the equation exposes the transit term, encompassing the first 100 simulation cycles of the task duration T_E. For each time step within this time interval and for each robot $r \in \mathcal{R}$, the fitness rises inversely with the Euclidean distance between the robot and the light position. At any time step, if the distance is larger than $\rho_\ell(t)$ then the fitness contribution at that time instant t will be zero (negative fitness scores are avoided using the $\max\{\cdot\}$ operator). The maximum distance to the light, $\rho_\ell(t)$, at which robots can still be rewarded is dependent on the time step t as shown in Eq. 4.4.

$$\rho_\ell(t) = \max\left\{1.5-\frac{t}{100},\,0.5\right\} \tag{4.4}$$

Essentially, the equation shows that its value starts as $\rho_\ell(0) = 1.5$ and then it decreases linearly as time elapses, up to a minimum value of 0.5. This lower bound of 0.5 is reached at the end of the transit phase ($t = 100$). In contrast, the steady state fitness term is shown in the second line of Eq. 4.3. It is a much more severe fitness evaluator because it directly verifies if the robots are within a ball of radius $\rho_\ell(\infty) = 0.5$ and centered at the light position. This validation is accomplished using the Heaviside function $\mathcal{H}$, that returns 1 if its input is greater than zero, or zero otherwise. Thus, only those robots whose distance to the light is lower than 0.5 will contribute to the fitness score rise.

- **Cube transport fitness function**: As mentioned in the previous section, the task of transporting cubes consists in carrying blue cubes distributed along the arena and gathering them into the correct ground area. There are three ground areas in total, and the one with a yellow light above is the nest where the robots have to transport the cubes. We denote this correct ground area as g, its center coordinates as $\mathbf{x}_g$ and its radius as ρ_g. With this notation in mind, Eq. 4.5 computes the number of cubes correctly placed in the nest g at the end of the task at $t = T_E/2$.

$$n_c = \sum_{c\in\mathcal{C}}\mathcal{H}\left(\rho_g-\|\mathbf{x}_c(t_{end})-\mathbf{x}_g\|_2\right) \tag{4.5}$$

$\mathcal{C}$ is the set of blue cubes in the arena and $\mathcal{H}$ is the Heaviside function. The last time instant of the task execution is denoted as t_{end} with the aim of clarifying the meaning of the formula as much as possible. Thereafter, using the previously computed value of n_c (cubes correctly collected), the overall fitness function of the task is displayed in Eq. 4.6.

$$F_2 = \max\left\{\frac{n_c+\sum_{c\in\mathcal{C}}d_c}{N_{cubes}},\,0\right\} \tag{4.6}$$

N_{cubes} denotes the total number of cubes in the arena. In addition, for every cube c, d_c rises the fitness score when the cubes are moved towards the nest.

$$d_c = \begin{cases} \min\left\{\frac{1}{1.5}\,\|\mathbf{x}_c(t_{end})-\mathbf{x}_c(0)\|_2\,,\,1\right\}, & \text{if } \|\mathbf{x}_c(t_{end})-\mathbf{x}_g\|_2 < \|\mathbf{x}_c(0)-\mathbf{x}_g\|_2 \\ 0, & \text{Otherwise} \end{cases} \tag{4.7}$$

The computation of d_c is displayed in Eq. 4.7, so that it is proportional to the Euclidean distance between the cube ending position, at t_{end}, and the position at the beginning of the task $t = 0$. Notice that if the cube has been displaced in the direction opposite to the ground area g, then $d_c = 0$. The motivation behind including the summation of distances $\sum_{c \in \mathcal{C}} d_c$ to the fitness function is to smooth and ease the evaluator, to such an extent that NEAT is able to find proper solutions more readily and avoid deceptive evolution trajectories. For instance, in a scenario in which robots are capable of suitably moving the cubes towards the correct nest but they are not able to reach the goal, the term of d_c would still reward the swarm behavior. Summarizing, the cubes transport fitness has two terms, that are normalized by the number of cubes. These terms account for the number of cubes transported and gathered into the correct nest and the sum of the distances that each cube has been moved (if the movement is in the correct direction).

- **Fitness function combination**: The previously defined fitness functions, which correspond to individual evaluations of the tasks, must be combined in order to generate the task switching overall fitness function. Thus, the total fitness score should be highest when the robots perform proficiently in both tasks. Our proposal is to use the geometric mean of the individual fitness values of the tasks. This computation is shown in Eq. 4.8 for our scenario with two tasks.

$$F_{tot} = \sqrt{F_1 \, F_2} \tag{4.8}$$

We avoided the use of the arithmetic mean because it can straightforwardly favour the optimization of only one of the tasks. On the contrary, the geometric mean avoids that situation because F_{tot} will only rise if both F_1 and F_2 are simultaneously optimized.

To conclude with this section, it should be mentioned that the genotypes are evaluated in multiple trials or episodes in order to reduce the estimation variance. Provided that each genotype is evaluated N_E times, the final fitness score estimate is the sample mean of the resulting fitness scores in each simulation trial. Clearly, the experiment environment and neuronal dynamics are reset and new world initialization is provided in each trial.

4.3 Experimental Setup

4.3.1 Neural Controller

The robot controller is the processing step of the robot execution responsible for accomplishing the mapping between the environment state, measured by the sensors, and the robot actions, to be fed to the actuator. In general, the controller can be any program or black box that performs the mentioned mapping. For instance, it can be implemented as a Finite State Machine (FSM), as a linear combination of states or based on artificial physics. Therefore, generally speaking, we refer to the robot controller as in Eq. 4.9,

$$\mathbf{a}(t) = \text{controller}\left(\boldsymbol{\phi}(t)\right) \tag{4.9}$$

where $\boldsymbol{\phi}(t)$ is the overall state vector, comprising the measurements of all the sensors at time step t. Aside from behavior-based controllers, in which the researcher has to design and code the entire controller and its rules, automatic design methods use optimization algorithms to find suitable controllers. The most popular types of optimized controllers are the neural controllers. Essentially, the neural controller harnesses the computational complexity and learning capabilities of artificial neural networks in order to build utterly effective and biologically plausible robot controllers. In this Master Thesis, neural controller composed by

CTRNNs, and optimized using NEAT and Hebbian learning, are used. The parameters of the CTRNN that are subject to the evolution process are the neuron decaying time constant, the neuron gain, the neuron bias, the synapses' weights and the parameters of the learning rule of each synapse (which are A, B, C and D). Moreover, owing to the fact that NEAT is applied, the topology of the CTRNN is also evolved. The CTRNN inputs and outputs depend on the experiment to be treated.

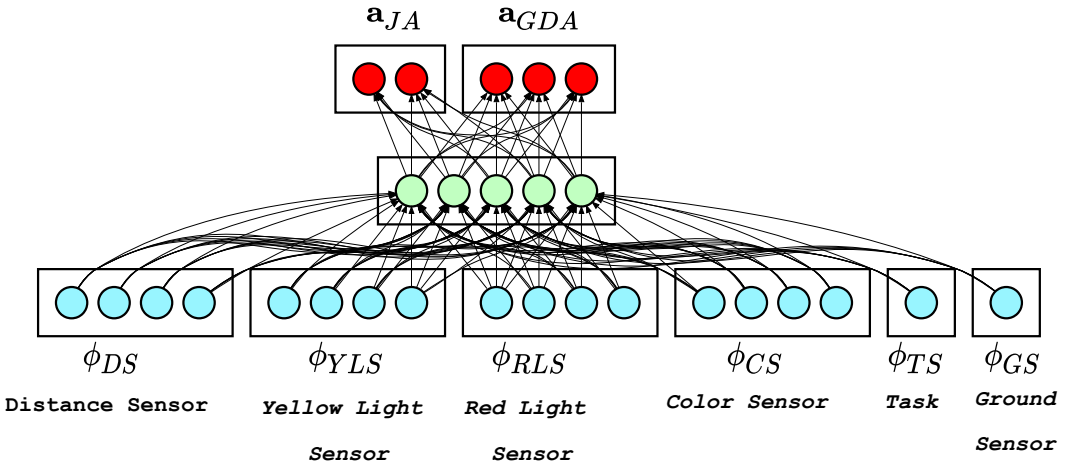


Figure 4.4: Initial CTRNN architecture of Experiment 1. The blue circles are the input nodes, the green circles represent the hidden neurons and the red dots are the motor neurons.

Fig. 4.4 exposes the initial CTRNN structure used in Experiment 1. The blue circles denote the input nodes, representing each component of the sensors' reading. Notice that each box or ensemble of nodes gathers the inputs of each sensor. The sensors of the first experiment are the distance sensor, the yellow light sensor, the red light sensor, the color sensor, the ground sensor and, finally, a binary input ϕ_{TS} representing the current task to be carried out. On the contrary, the motor neurons (in red) from where actions are decoded correspond to the joint action (a_{JA}) and the grasp and drop action (a_{GDA}). The activation of almost all the neurons is the sigmoid activation. The exceptions are the motor neurons controlling the joints of the robot, which have a hyperbolic tangent activation in order to allow reverse motion. Furthermore, the action a_{GDA} is decoded using a softmax function in order to select an action within the set $\{0, 1, 2\}$. Finally, NEAT is initialized with a hidden layer of 5 neurons. Notice that the CTRNN depicted in Fig. 4.4 is just the topology at the beginning of the evolution.

In the case of experiment 2, the initial architecture of the CTRNN controlling the robot actions is illustrated in Fig. 4.5. The most significant difference is the addition of the communication information. More precisely, the last input ensemble $\boldsymbol{\phi}_{RX}$ holds the perceived messages from each of the four sectors of the communication receiver. On the opposite side, the motor layer with one neuron generates the action message to be broadcasted a_{TX}. Additionally, two initial hidden layers are established. The first one receives the readings from the distance sensor and the two light sensors while the second hidden ensemble receives the messages from the communication receiver. The binary action ϕ_{TS}, with the current task to be accomplished, is provided to both hidden layers.

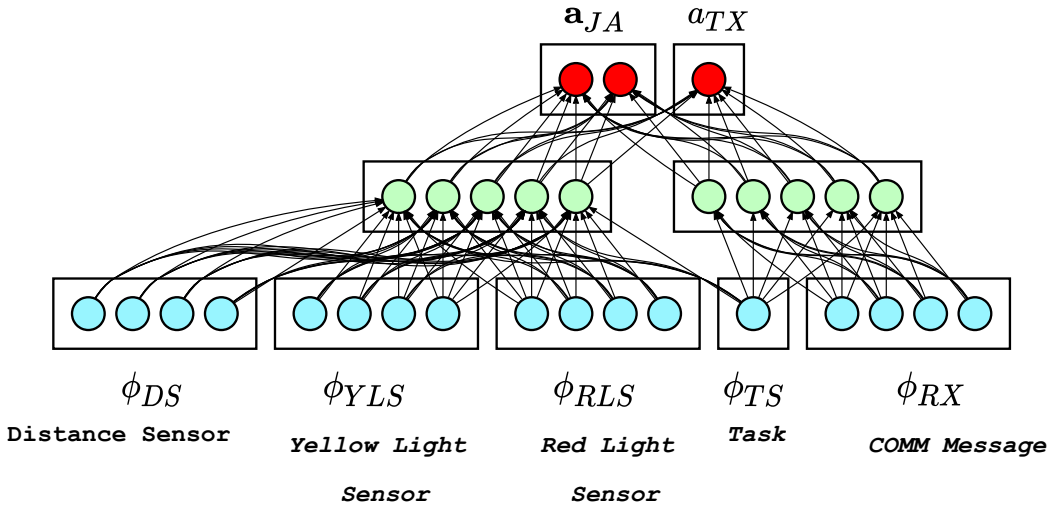


Figure 4.5: Initial CTRNN architecture of Experiment 2. The blue circles are the input nodes, the green circles represent the hidden neurons and the red dots are the motor neurons.

4.3.2 Evolution Setup

As previously stated, the parameters and topology of the CTRNNs are evolved using NEAT algorithm. Moreover, the weights of the synapses are adapted online by means of generalized ABCD Hebbian learning rules. The learned weights at the end of the simulations are not transmitted to the next generation. The learning experience is indirectly transmitted to the evolution process with the resulting fitness score achieved in the episode. An important aspect to be treated is how the genotype to phenotype mapping is accomplished. As briefly mentioned in Sec. 3.1.3, the genotypes, which are lists of connection and neuron genes, are converted to a CTRNN (phenotype) by constructing the corresponding graph $\mathcal{G}_{\text{RNN}}$. In the search space of NEAT, every neuron and synapse parameter is constrained to the interval [0, 1]. Thus, in order to accomplish this normalization, the evolvable variables are subject to the transformations shown in Eq. 4.10.

$$\left.\begin{aligned} \hat{w}_{ij} &= \frac{w_{ij} - w_{min}}{w_{max} - w_{min}} \\ \hat{g}_i &= \frac{g_i - g_{min}}{g_{max} - g_{min}} \\ \hat{\beta}_i &= \frac{\beta_i - \beta_{min}}{\beta_{max} - \beta_{min}} \\ \hat{\tau}_i &= \frac{\log(0.5\,\tau_i) - \tau_{min}}{\tau_{max} - \tau_{min}} \end{aligned}\right\} \tag{4.10}$$

Where, for every connection and every neuron, $\hat{w}_{ij}$, $\hat{g}_i$, $\hat{\beta}_i$, $\hat{\tau}_i \in [0, 1]$ are the normalized parameters. Additionally, the maximum and minimum values of each variable in the phenotype domain will be specified below. The normalization of the parameters is linear in all the variables, excluding the neuron decaying time constants. The parameters of the learning rule, A, B, C and D, are also subject to the same linear transformation as, for

instance, in the weight mapping of Eq. 4.10.

$$\left.\begin{aligned} w_{ij} &= \hat{w}_{ij}\,(w_{max} - w_{min}) + w_{min}, \\ g_i &= \hat{g}_i\,(g_{max} - g_{min}) + g_{min} \\ \beta_i &= \hat{\beta}_i\,(\beta_{max} - \beta_{min}) + \beta_{min}, \\ \tau_i &= 2 \cdot 10^{\hat{\tau}_i\,(\tau_{max} - \tau_{min}) + \tau_{min}} \end{aligned}\right\} \tag{4.11}$$

Variable	Min.	Max.
$\mathbf{w}$	-5	5
$\boldsymbol{\tau}$	-0.8	1.2
$\boldsymbol{\beta}$	-1.5	1.5
$\mathbf{g}$	0.05	5
A, B, C, D	-2	2

Table 4.1: Search space constrains of each of the denormalized optimization variables.

The opposite operation, shown in Eq. 4.11, denormalizes the search space [0, 1] used by NEAT into the CTRNN parameter space bounded by the maximum and minimum values defined by the researcher. Table. 4.1 collects the maximum and minimum values of all the variables used in Eqs. 4.10 and 4.11. Firstly, notice that due to the non-linear normalization, the $[-0.8, 1.2]$ range of $\boldsymbol{\tau}$ is mapped to $[2 \cdot 10^{-0.8}, 2 \cdot 10^{1.2}] = [0.32, 31.7]$ seconds. This means that, provided that the CTRNN Euler step Δt is set to 0.1, the decaying time constants are bounded by $[3.2\,\Delta t, 317\,\Delta t]$. This search space of $\boldsymbol{\tau}$ 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 enable the formation of working memory at a much slower time scale. Besides, we heuristically found the shown bounds of the weights and biases to be a good tradeoff between strong amplification and saturation avoidance. The gain values are restricted uniquely to positive values, avoiding the scenario of zero gain.

Lastly, Table 4.2 gathers the hyperparameters used in the evolution process of NEAT. These values are mainly obtained through heuristic search, trial and error and also considering the experimental setup proposed by the authors of NEAT in their experiments (see [88]). The table also shows a brief description of each variable. The population size is set to 300 and for each population genotype, the fitness is evaluated 10 times. This configuration would be utterly costly unless parallelization is leveraged (as in our case). Provided that λ_i is the size of species i, the constant n_{sel} is the number of genotypes selected in each species as parents. As explained in Chapter 2, this election is accomplished using the tournament selection. Besides, the probability of node mutation is lower than the probability of adding new connections in order to avoid undesired fast topology growth. The constants c_1 and c_2 are fixed to the same value because the same importance is given to both excess and disjoint gene differences. The similarity threshold δ_t was probably the most challenging hyperparameter to be adjusted. We found $\delta_t = 0.3$ to be a nice tradeoff for the used CTRNNs. The elites, as stated in the corresponding description, maintain untouched 2 individuals from each species. To conclude, η is the learning rate of the Hebbian learning rules, which is the same for all the synapses.

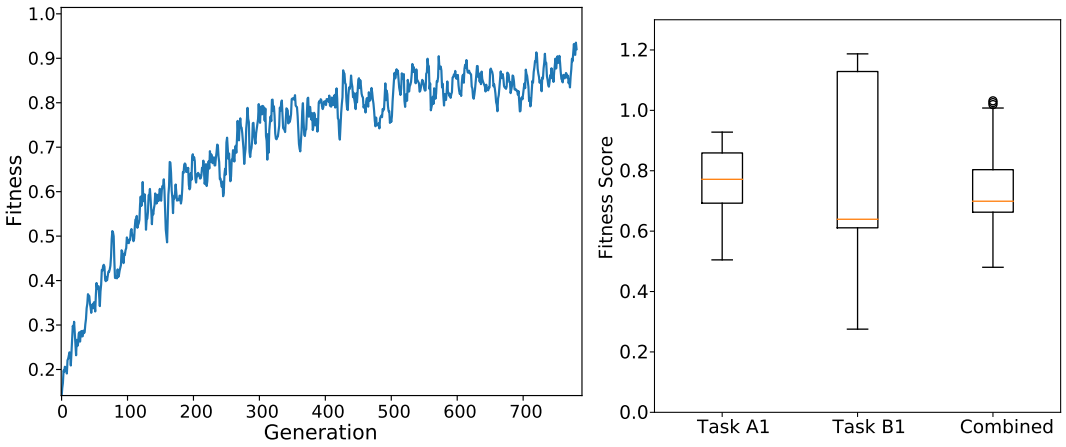
Variable	Value	Description
λ	300	Population size.
N_E	10	Number of genotype evaluations.
n_{sel}	$0.3\,\lambda_i$	Number of parents selected for crossover in species i.
p_{mw}	0.1	Probability to modify gene parameters.
p_{mc}	0.15	Probability to add a new connection.
p_{mn}	0.05	Probability to add a new neuron.
σ_w	0.1	Std. dev. of the gaussian parameter mutation.
c_1, c_2	1	Importance of the disjoint and excess terms when computing the similarity between two genotypes.
c_3	0.6	Importance of the parameter distance when computing the similarity between two genotypes.
δ_t	0.3	Similarity threshold of speciation.
E	2	Number of elites per species
η	$5 \cdot 10^{-3}$	Learning rate of the Hebbian learning rules.

Table 4.2: Best found evolution hyperparameters for all the experiments

Chapter 5

Results

This chapter presents the results of the experiments defined in Chapter 4, once the evolution process has ended. For each experiment, aside from illustrating and discussing the achieved fitness scores, the best performing genotype (in terms of fitness value) is selected and post evaluated. The postevaluation analysis mainly focusses on the robots' behavior, performance and proficiency against the proposed task switching experiments. Furthermore, in the results of Experiment 2, the communication semantics and mechanics that emerge from NEAT evolution are described. In order to construct all the graphics, we use a statistically significant sample comprising 50 different and independent simulation evaluations of the genotype. Using this sample, most of the figures show statistics instead of isolated trajectories. Sec. 5.1 is devoted to the presentation of the results obtained in Experiment 1 and Sec. 5.2 gathers and displays all the outcomes of Experiment 2. The behavior and performance of the robots in both experiments can be visualized in video format at `https://www.youtube.com/watch?v=G8MvWT5ATRQ`.

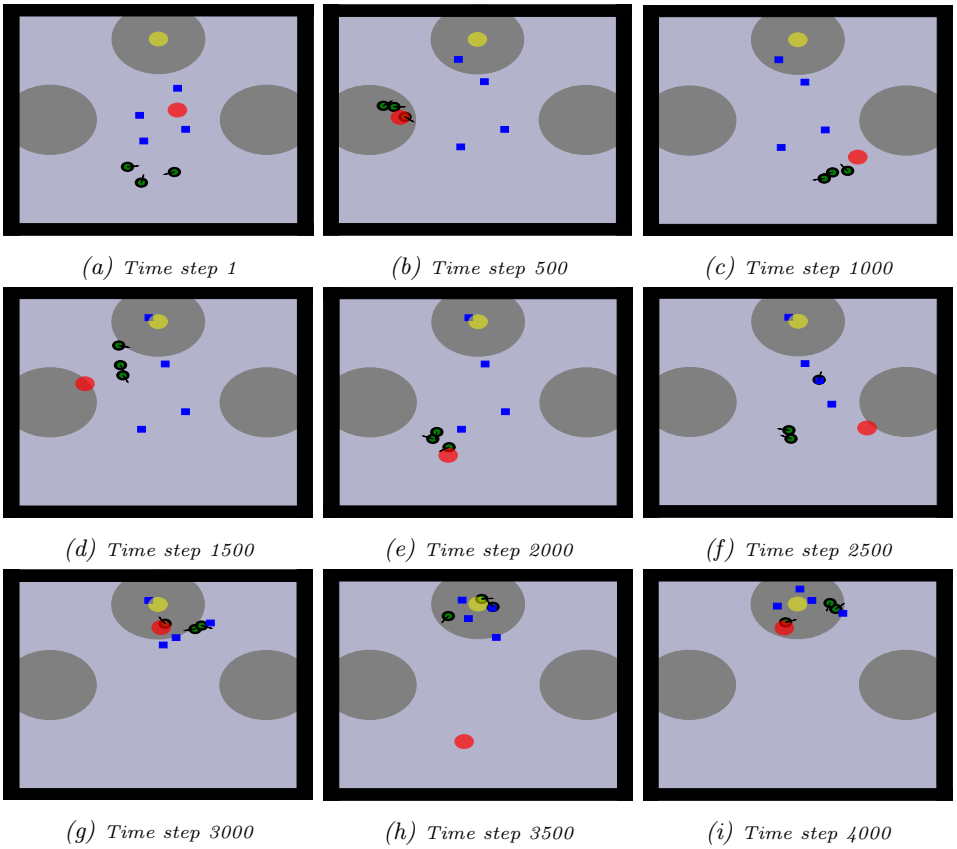


(a) *(b)*

Figure 5.1: (a) Fitness score evolution as generations are elapsed. For each generation it shows the maximum fitness of the population. (b) Boxplots illustrating the distribution of the fitness score of the best individual after evolution. The boxes correspond to the fitness scores resulting from a post evaluation of the fittest genotype 50 independent times. The boxplots expose the fitness values of the task A1, task B1 and their combination as a geometric mean. Within the boxplots, the orange lines within the boxes are the median values 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.

5.1 Experiment 1

As established in Sec. 4.1.1, Experiment 1 proposes a task switching problem in which robots have to learn the tasks of following a mobile light source (Task A1) and transporting small cubes to a common nest area (Task B1). Firstly, Fig. 5.1 illustrates the performance in terms of achieved fitness score. Fig. 5.1a shows the evolution of the fitness value as generations of NEAT are elapsed. Specifically, for each generation it plots the fitness of the best performing genotype in the population. In contrast, in Fig. 5.1b, a post evaluation analysis of the fitness of the best individual is depicted. After the NEAT evolution process, the fitness distribution of the best genotype is represented as boxplots using a 50 fitness score samples of independent evaluation trials. Furthermore, the boxplots are broken down into the fitness of the Task A1, Task B1 and the aggregation of both (see Sec. 4.2). Notice that the interquantile range of the boxplot corresponding to Task B1 is remarkably large. The underlying reason is that a slight variation in the behavior of the robots can lead to an abrupt rise or decrease of the fitness score. In other words, the fitness function in Eq. 4.6 is discontinuous because it depends on the number of cubes correctly collected (n_c). Provided that there are 3 cubes in the arena, the fitness difference between collecting 2 or 3 cubes is $\Delta F_2 = 1/N_{cubes} = 0.33$.



(a) Time step 1 *(b) Time step 500* *(c) Time step 1000*

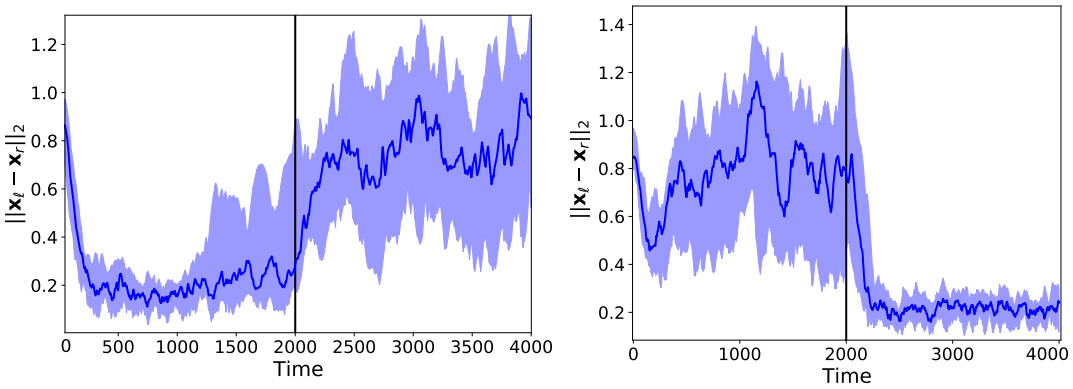
(d) Time step 1500 *(e) Time step 2000* *(f) Time step 2500*

(g) Time step 3000 *(h) Time step 3500* *(i) Time step 4000*

Figure 5.2: Frames of a simulation of experiment 1 under the task order A1, B1. The switch of tasks is produced at time step 2000. Notice that, in order to ease visualization, the shown frames are not actual snapshots of the simulator but recreated 2D plots using the recorded data. The robots are represented using green balls with black contour and with a line denoting its heading orientation. The cubes are the blue squares, the red and yellow lights are painted as circles of the corresponding color and the large grey circles are the ground areas.

With these discontinuities in mind, the median value of the fitness score of Task B1 around 0.66 means that, in median, the robots are able to collect 2 cubes out of 3. Besides, the fitness scores of Task A1 are remarkably outstanding, considering that the maximum fitness of 1 is almost impossible to be achieved in practice. Lastly, the combined fitness distribution indicates the proposed task switching and sequential multitasking problem is successfully solved.

Aside from showing the performance in terms of achieved fitness scores, Fig. 5.2 exposes different frames of a simulation of Experiment 1 using the best evolved genotype. In the sample simulation, the robots have to start with Task A1 and conclude with Task B1. The task switching is requested at simulation cycle 2000. In the first 5 frames (from (a) to (e)), it can be observed that the swarm members successfully follow the red light source. In contrast, from Fig. 5.2f until the ending of the simulation, the robots ignore the red light source and aim at transporting the blue cubes to the ground area in the north. The last frame illustrates that all the cubes have been suitably collected in the correct ground area.



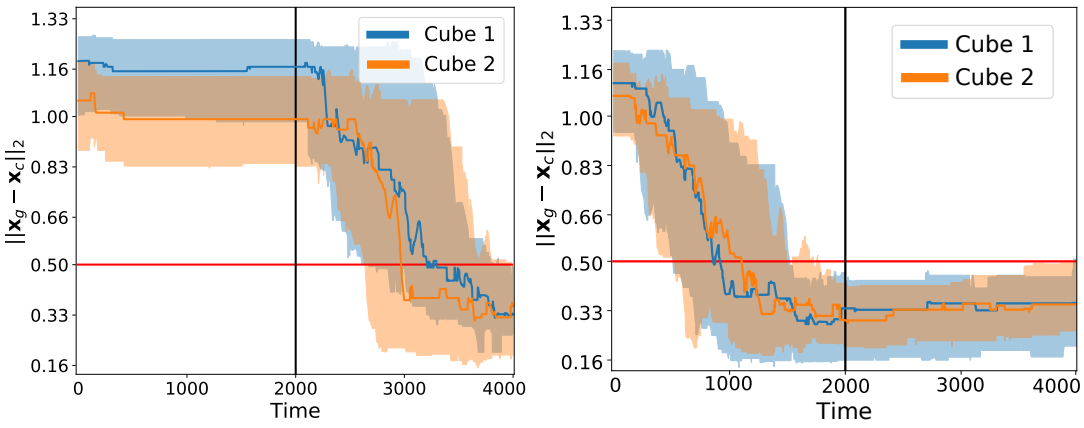
(a) Task Order: A1 ⟶ B1

(b) Task Order: B1 ⟶ A1

Figure 5.3: Temporal evolution of the Euclidean distance between robots and the red light source under different orderings of tasks. In (a) Task A1 is presented first and in (b) Task B1 is the starting task. In both cases the switch of task is produced at time instant 2000, which is represented with a vertical line. The dark blue curves represent the median value of the distance using the robots positions of 50 independent simulations. The contours of the blue shadows illustrate the first (lower shadow) and third (upper shadow) quantiles.

Focussing on the behavior of the swarm against Task A1, Fig. 5.3 illustrates the distance between robots and the targeted red light source, as simulation cycles are elapsed. In order to build the plots, 50 independent simulation episodes are collected. The dark blue curve represents the median value, among the 50 sample simulations, of the Euclidean distance between the robot and the red light. Complementarily, the contours of the blue shadows depict the first and third quantiles. In Fig. 5.3a, the task order in A1, B1 and in Fig. 5.3b the ordering is B1, A1. Along this chapter, the reason underlying the use of both task sequences is to verify whether the solution provided by NEAT can generalize properly to different task orderings or not. The task switch is produced at simulation cycle 2000 and it is represented as a vertical line. In these two plots, it can be observed that the Euclidean distance drastically decreases when the requested task is A1. Furthermore, it reaches a steady state of about 0.2m of distance to the light, in median value. The distance variation when robots solve Task A1 is, generally, remarkably reduced. Regarding the time slots corresponding to Task B1, the distance to the light is larger and has much more variability because robots ignore the red light in order to collect blue cubes. Both plots expose that, at time step 2000, when the

task switching occurs, the robots almost instantaneously realize that the task to be solved has changed and they correctly modify their behavior.



(a) Task Order: A1 ⟶ B1

(b) Task Order: B1 ⟶ A1

Figure 5.4: Temporal evolution of the Euclidean distance between each instantiated cube and nest ground area, under different orderings of tasks. In (a) Task A1 is presented first and in (b) Task B1 is the starting task. In both cases the switch of task is produced at time instant 2000, which is represented with a vertical line. The horizontal red line indicates the radius of the ground area. The dark curves represent the median value of the distance using the cubes and nest positions of 50 independent simulations. The contours of the shadows illustrate the first (lower shadow) and third (upper shadow) quantiles.

Thereafter, Fig. 5.4 depicts the behavior of the robots against Task B1, in which the swarm has to collect the cubes. As an example, a scenario with 2 cubes is used to collect the data that is shown in the plots. For each cube, the figure displays the Euclidean distance between the cube's position and the center of the nest where objects have to be stored. Recall from Sec. 4.1.1 that the correct nest is the one underneath the yellow light source acting as beacon. The orange and blue dark curves indicate the temporal evolution of the median value of the previously mentioned distance using 50 independent simulation trials. Complementarily, the contours of the shadows highlight the temporal evolution of the first and third quantiles. Fig. 5.4a shows the results when the task order is A1, B1 while Fig. 5.4b exposes the distance time series when Task B1 is firstly requested. The vertical black line marks the time instant when the task switching is produced and the horizontal red line indicates the radius of the nest ground area. Therefore, when the distance between cubes and the nest is below this threshold, it means that the corresponding cube has been properly stored inside the correct nest. It can be appreciated that, at the last cycle of the corresponding time slot (4000 in Fig. 5.4a and 2000 in Fig. 5.4b), almost all the cubes are correctly placed inside of the nest. The rate at which robots displace the cubes is also remarkably steady along the 50 simulations. Furthermore, within the time window corresponding to Task A1, the cubes are generally static because the robots are accomplishing the red light pursuit. Additionally, Fig. 5.5 breaks down, out of a total of 50 simulations, the percentage of simulations that the robots were capable of collecting from 0 up to 5 cubes. The experiment is accomplished with 2 robots and 5 cubes. It can be observed that the most common scenario is the one in which robots correctly recollect 3 out of 5 cubes in total (about 33%). It is also highly usual the case in which only 2 cubes are collected, corresponding to each robot suitably carrying a single cube. Nonetheless, it is also remarkably recurrent the outstanding scenario in which the cubes transportation task is completely solved (5 out of 5 cubes collected), in about 14% of the simulations. The situations

with the robots performing poorly are drastically minimized. For instance, the mobile robots are unable to collect any cube merely in 4% of the simulations.

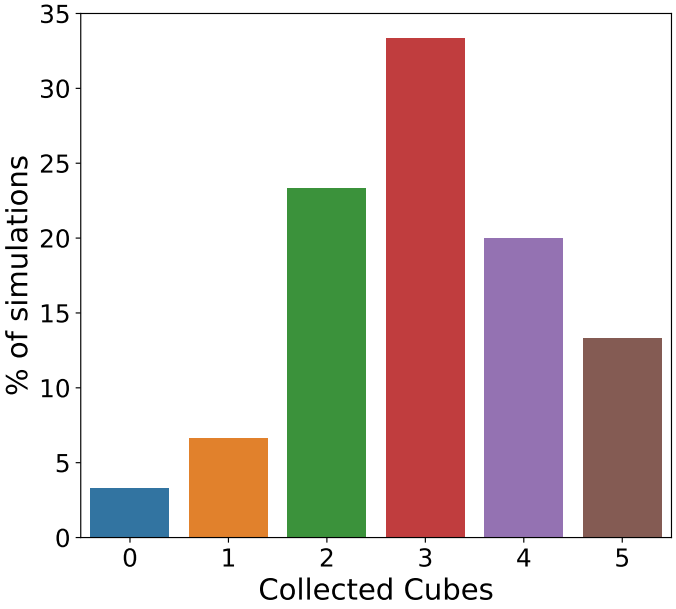
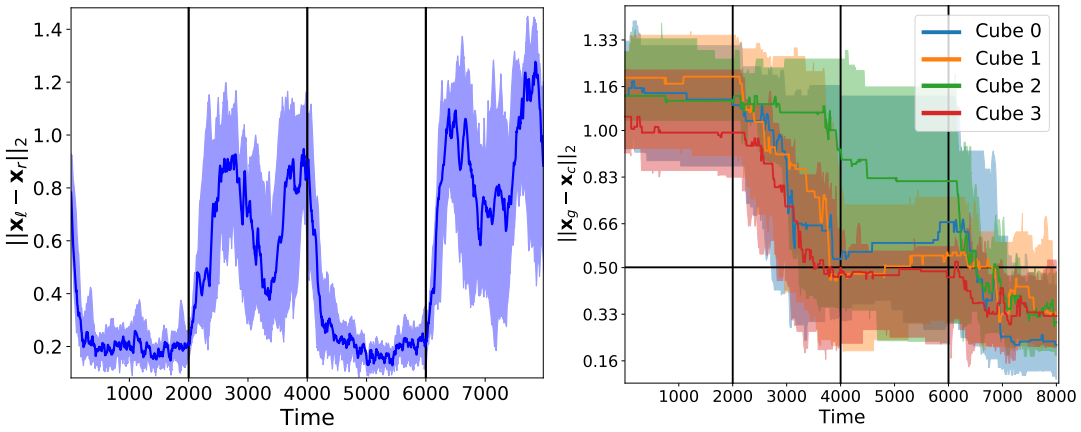


Figure 5.5: Barplot showing the percentage of independent simulations in which the robots were capable of correctly collecting any, 1, 2, 3, 4 or 5 cubes.

The previous figures have displayed the results when there is only one task switching instant within the experiment duration. This configuration is the only one that has been requested to the swarm during the evolution process. However, it is utterly interesting to validate if the optimized CTRNN controllers are able to generalize to multiple changes between tasks. More precisely, the simulation is split into 4 time slots with task sequence A1 $\longrightarrow$ B1 $\longrightarrow$ A1 $\longrightarrow$ B1. Fig. 5.6 shows the results in this new scenario. Firstly, in Fig. 5.6a it is illustrated the Euclidean distance to the red light, showing the median value and the quantiles. In this case, as there are 4 task time slots, the number of times that robots have to switch between tasks is 3 (highlighted with vertical lines). The distances within the pertaining time slots 1 and 3 are suitably reduced, showing that the task switching can be correctly accomplished with more than two time slots. In contrast, in Fig. 5.6b, the distance between the cubes and the nest center are shown. In order to hinder the problem with respect to Fig. 5.4, a total of 4 cubes are instantiated in this case. The plot verifies that the robots successfully transport the cubes at the correct time slots 2 and 4. In the time windows 1 and 3 the cubes are static because the swarm is following the red light source. Very interestingly, at time step 4000, only two cubes are correctly placed within the nest, in median value. It is not until the last time slot (from time step 6000) that robots manage to store the remaining cubes, corresponding to the the green and blue curves in the figure, within the nest ground area.



(a) Euclidean distance between the robots' position to the red light.

(b) Euclidean distance between the cubes' position to the center of the nest area.

Figure 5.6: Results of Experiment 1 with 4 time slots and a duration of 8000 time instants. The changes between tasks A1 and B1, highlighted by vertical lines, are produced at simulation cycles 2000, 4000 and 6000 and the task sequence is {A1, B1, A1, B1}. (a) Euclidean distance between robots and the red light source. (b) Euclidean distance between the cubes and the center of the ground area underneath the yellow light. In both plots, 50 independent simulations are collected and the dark curves represent the median distance, while the shadow contours of each time series corresponds to the first and third quantiles.

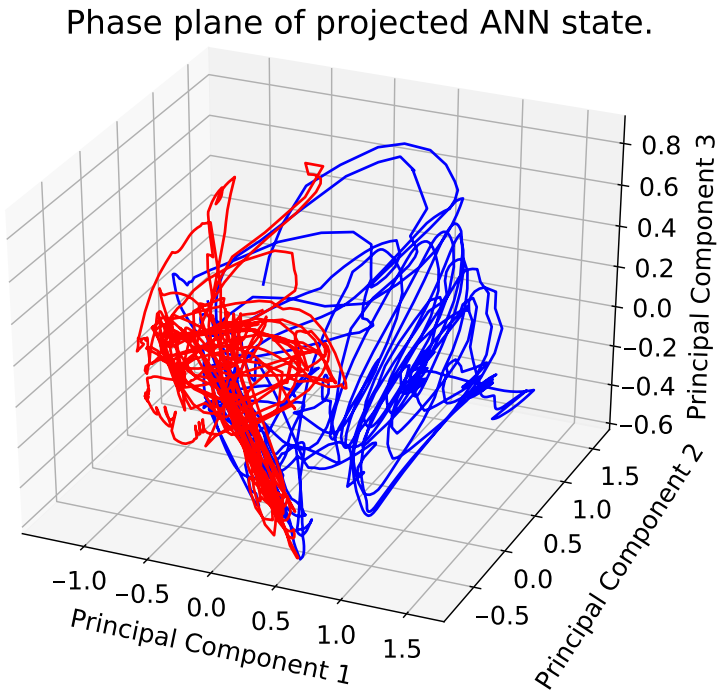


Figure 5.7: CTRNN state space trajectory of Experiment 1. It shows the trajectory of the neurons' firing rates projected onto 3 dimensions using PCA. The considered trajectory lasts 4000 simulation cycles and the blue curve corresponds to the time instants of Task A1 while the red curve depicts the time instants when Task B1 is addressed.

To conclude the presentation of the results obtained in Experiment 1, the state space trajectories of the CTRNN are explored. Specifically, the dynamics of the CTRNN are observed in terms of the firing rates of the neurons, $\mathbf{u}(t)$. However, owing to the fact that the dimension of the state space is $N >> 3$, we apply principal components analysis to the recorded CTRNN dynamics simulation. Principal Components Analysis (PCA), see [102, 103], is a popular dimensionality reduction algorithm that uses a linear transformation of the data to project it onto a lower dimensional space (in our case a 3D space). Even though it is not its main use, we employ PCA merely for visualization purposes (in a similar way as in [104]). Fig. 5.7 illustrates a sample trajectory of the projected neurons' firing rates $\mathbf{u}(t)$ when Experiment 1 is addressed. The exposed trajectory comprises a total of 4000 time steps. Moreover, the blue curve represents the first 2000 time instants, when the robots are addressing the Task A1. In contrast, the red curve highlights the trajectory between time steps 2000 and 4000, when Task B1 is solved. The most interesting aspect to be pointed out in the figure is that depending on the task being faced, the CTRNN's dynamics are driven towards a different part of the state space. This fact is clearly observable in Fig. 5.7, where the state space trajectories corresponding to each task are straightforwardly separable.

5.2 Experiment 2

As described in Sec. 4.1.2, the second experiment is devoted to the study and analysis of the communication emergence through artificial evolution. Moreover, the communication is merged with the task switching already treated in the previous section, albeit the tasks are not the same. Let us recall that in Experiment 2 the tasks are called Task A2 and Task B2 and consist of following mobile red and yellow light sources, respectively. In order to correctly solve the experiment, some sort of communication has to arise, because only one robot of the group actually knows which task has to be executed at each time instant. Firstly, the fitness

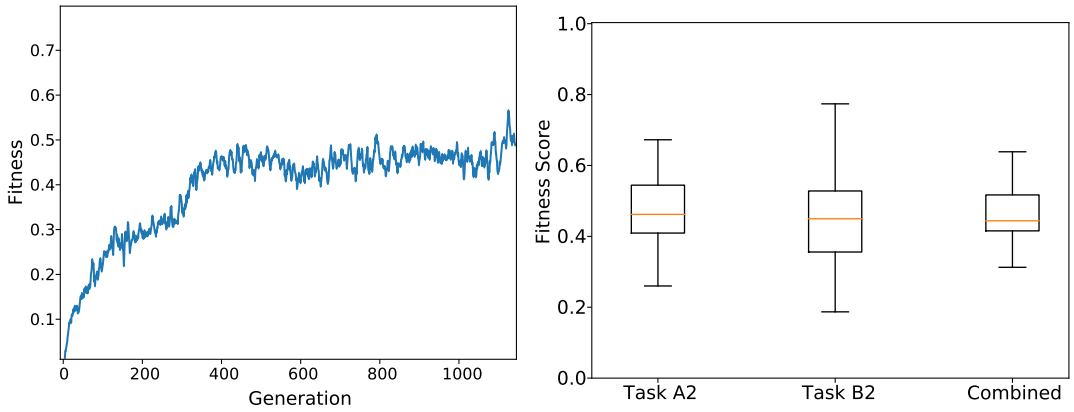
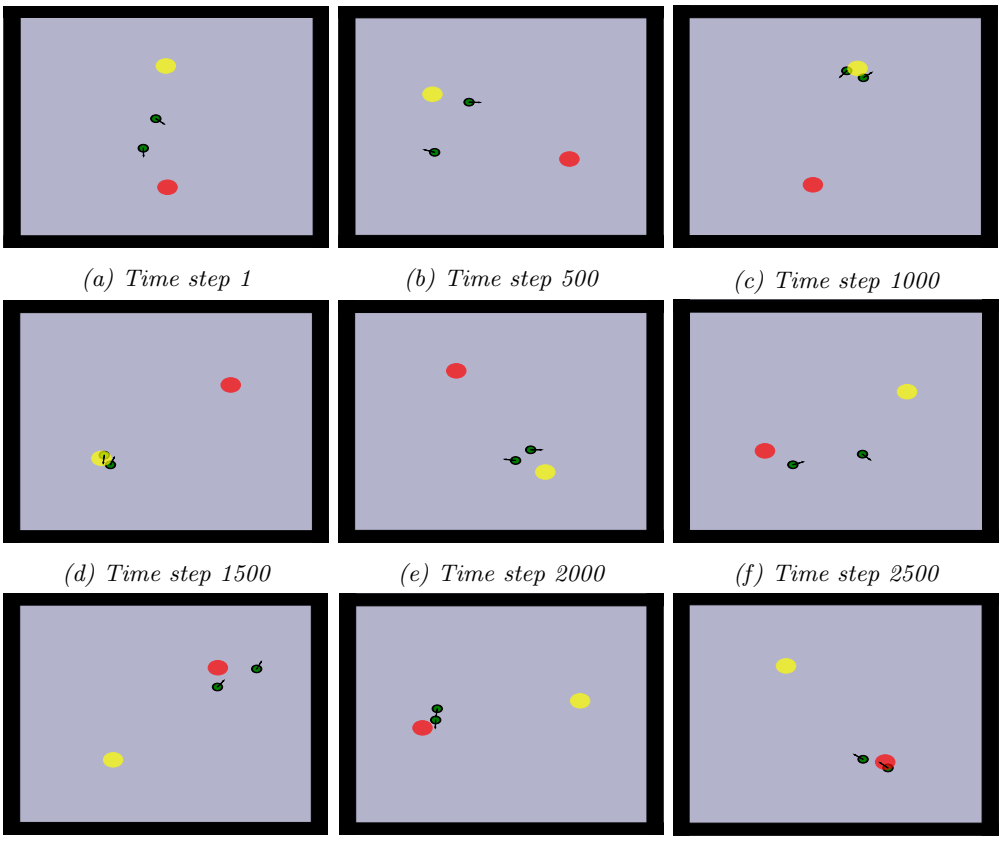


Figure 5.8: (a) Fitness score evolution as generations are elapsed in Experiment 2. For each generation it shows the maximum fitness of the population. (b) Boxplots illustrating the distribution of the fitness score of the best individual after evolution. The boxes correspond to the fitness scores resulting from a post evaluation of the fittest genotype 50 independent times. The boxplots expose the fitness values of the task A2, task B2 and their combination as a geometric mean. Within the boxplots, the orange lines within the boxes are the median values 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.

function achieved by the robots in this experiment is illustrated in Fig. 5.8. Fig. 5.8a shows the evolution of the maximum fitness score within the NEAT population as generations are elapsed. Notice that the abrupt fitness rise at the begging of the evolution might represent that robots learn to solve only one of the two tasks straightforwardly, albeit without any kind of communication. Moreover, even though it is not verified, our hypothesis is that the communication emerges at the second fitness score rise, approximately at generation 300. Complementarily, a postevaluation analysis of the fitness value distribution is illustrated in Fig. 5.8b by means of boxplots. The analysis is performed by evaluating 50 times the best genotype, after the evolution process has concluded. Thereafter, the recoded fitness score samples are jointly plotted within the boxplots, breaking down the distributions of the individual tasks and the aggregation using the geometric mean. Although the fitness scores achieved and presented in Fig. 5.8 still have margin of improvement, it should be remarked that the absolute maximum achievable fitness of 1 is nearly impossible in practice. Furthermore, as it will be demonstrated in the remaining of this section, the reached robot behavior is sufficiently good in fulfilling the experiment requirements. The behavior of the robots in Experiment 2 can be appreciated in Fig. 5.9, which displays different frames of



(a) Time step 1 *(b) Time step 500* *(c) Time step 1000*

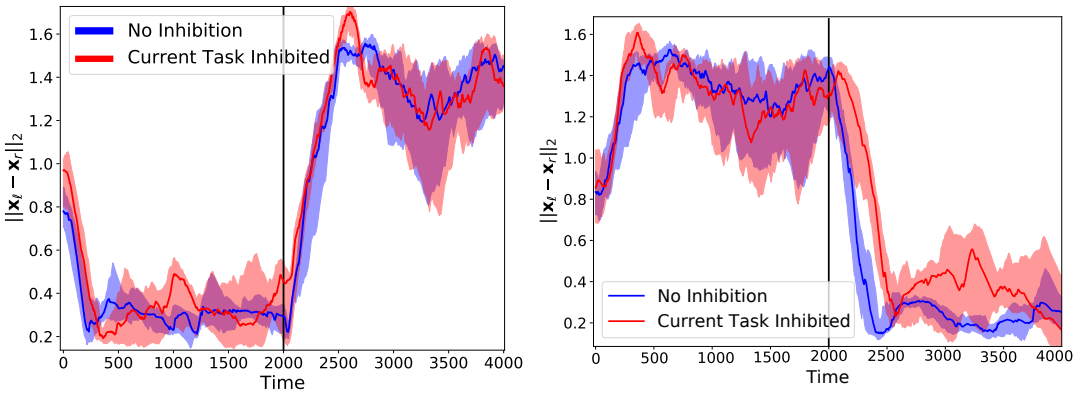
(d) Time step 1500 *(e) Time step 2000* *(f) Time step 2500*

(g) Time step 3000 *(h) Time step 3500* *(i) Time step 4000*

Figure 5.9: Frames of a simulation of experiment 2 under the task order A2, B2. The switch of tasks is produced at time step 2000. Notice that, in order to ease visualization, the shown frames are not actual snapshots of the simulator but recreated 2D plots using the recorded data. The robots are represented using green balls with black contour and with a line denoting its heading orientation. The red and yellow lights are painted as circles of the corresponding color.

a sample simulation that starts with Task B2 (yellow light) and concludes with Task A2 (red light). It can be observed that, until time instant 2000, the robots successfully pursuit the yellow light closely. Similarly, from time step 2000 until the end of the simulation, the robots are capable of proficiently switching between tasks and following the red light source. Moreover, although only one robot knows the task to be solved, it is important to remark that all the robots of the swarm are able to solve both tasks.

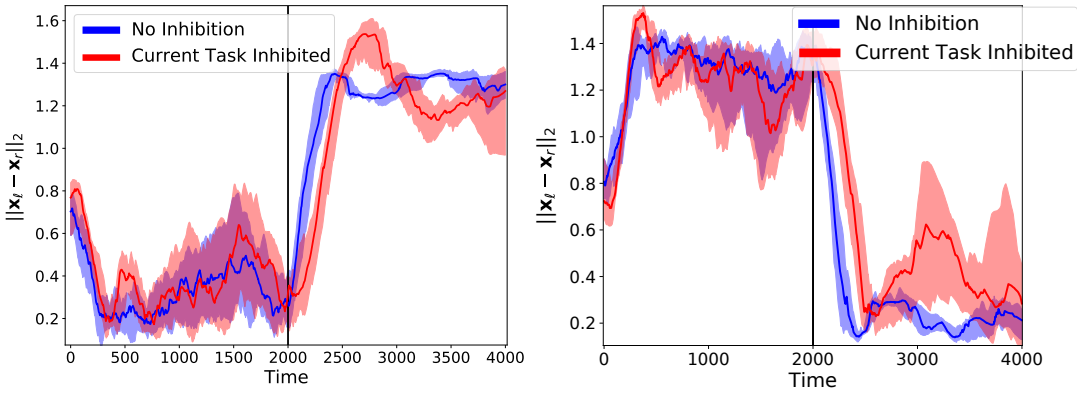
In contrast to Fig. 5.9, which shows a single evaluation, that could have been biased, Fig. 5.10 and 5.11 illustrate the results using a statistically significant sample of 50 independent simulations. Firstly, Fig. 5.10 shows the temporal evolution of the Euclidean distance of robots to the red light source. The blue and red dark curves represent the evolution of the median value of all the distances and the upper and lower bounds of the shadows denote the first and third quantiles, respectively. Moreover, the blue lines of both plots represent the distance to the light of the robots that are unable to know the current task. On the contrary, the red curves are constructed using only those robots that have access to the task information. In Fig. 5.10a, the distances are presented using the task sequence A2, B2. In contrast, Fig. 5.10b exposes the results when task B2 is requested first. The figures clearly display the successful performance of the swarm against Task A2 (red light pursuit),



(a) Task Order: A2 ⟶ B2 *(b) Task Order: B2 ⟶ A2*

Figure 5.10: Temporal evolution of the Euclidean distance between robots and the red light source under different orderings of tasks. In (a) Task A2 is presented first and in (b) Task B2 is the starting task. In both cases the switch of task is produced at time instant 2000, which is represented with a vertical line. The dark blue curves represent the median value of the distance using the robots positions of 50 independent simulations. The contours of the blue shadows illustrate the first (lower shadow) and third (upper shadow) quantiles.

highlighting the fact that the distribution of robot distances is minimized at the correct time slots. Additionally, an interesting issue to be pointed out is that both red and blue curves drastically tend to decrease when Task A2 has to be solved. This means that some kind of inter robot communication has emerged from evolution. Furthermore, at the time windows corresponding to the other task B2, the distances to the red light settle to 1.5 meters. As it will be presented below, this means that the robots are also able to solve proficiently the remaining task B2 at the correct time slots.

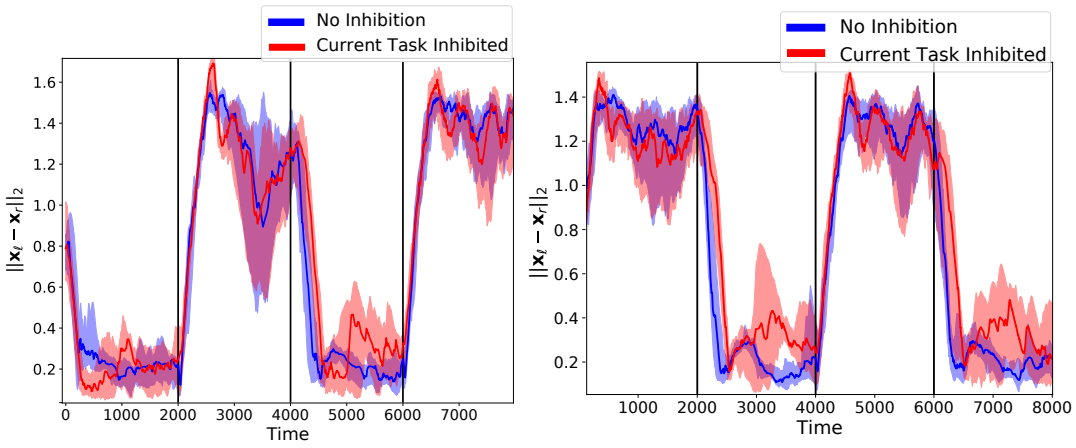


(a) Task Order: A2 ⟶ B2 *(b) Task Order: B2 ⟶ A2*

Figure 5.11: Temporal evolution of the Euclidean distance between robots and the yellow light source under different orderings of tasks. In (a) Task A2 is presented first and in (b) Task B2 is the starting task. In both cases the switch of task is produced at time instant 2000, which is represented with a vertical line. The dark blue curves represent the median value of the distance using the robots positions of 50 independent simulations. The contours of the blue shadows illustrate the first (lower shadow) and third (upper shadow) quantiles.

Fig. 5.11 complements the results of Fig. 5.10 showing the distances of the robots to the

yellow light, corresponding to the Task B2. In a similar way, the red curve comprises the distances of robots capable of acquiring the current task while the blue curve represents those robots without any task information. In the light of the results, it can be also stated that the robots are able to correctly solve Task B2 at the precise timing and using some sort of communication mechanics. As a final remark to Figs. 5.10 and 5.11, it is also interesting to analyze the transit period when the task switching occurs. This transit period essentially means that robots have to realize and process the change of task and, thereafter, they have to move to the antipode of the lights' orbit. In the figures, this transit period approximately lasts 500 time steps (between time step 2000 and 2500). Moreover, the blue curves, corresponding to those robots that know which light has to be sought, generally rise or decay firstly. On the contrary, the robots without the task information merely rely on the communication mechanics and, consequently, their rotation is delayed. Before entering into the emerged



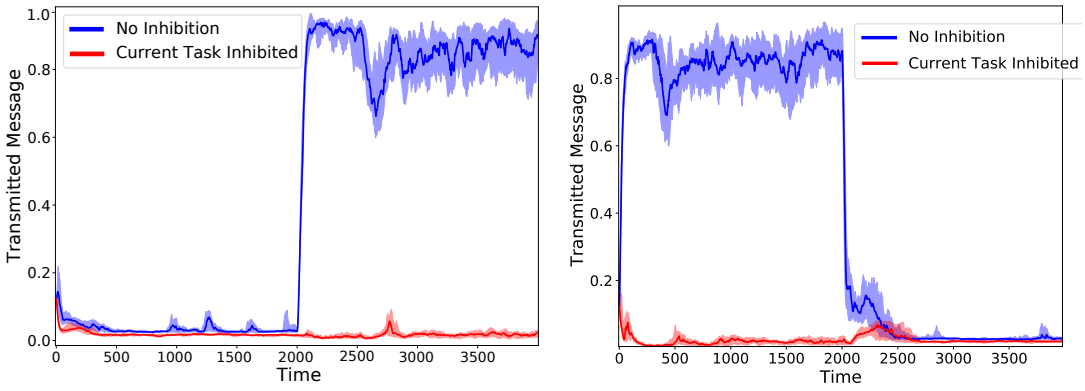
(a) Euclidean distance between the robots' position to the red light.

(b) Euclidean distance between the robots' position to the yellow light.

Figure 5.12: Results of Experiment 2 with 4 time slots and a duration of 8000 time instants. The changes between tasks A2 and B2, highlighted by vertical lines, are produced at simulation cycles 2000, 4000 and 6000 and the task sequence is {A2, B2, A2, B2}. (a) Euclidean distance between robots and the red light source. (b) Euclidean distance between robots and the yellow light source. In both plots, 50 independent simulations are collected and the dark curves represent the median distance, while the shadow contours of each time series corresponds to the first and third quantiles.

communication description, a final validation is accomplished in order to test the solution's robustness. Fig. 5.12 extends the previously shown figures to a sequence of 4 time slots and a duration of 8000 cycles. The task sequence is {A2, B2, A2, B2}, so that robots have to perform task switching in 3 different time steps. Fig. 5.12a shows the results of this evaluation against Task A2, depicting the distances to the red light. Similarly, Fig. 5.12b illustrates the distances to the yellow light source. In both cases, the results verify that the task switching is successfully achieved when multiple changes of labor are requested. Furthermore, even thought the former graphics suggest that the communication has correctly emerged at some extent, the precise mechanics are presented in the following.

Fig. 5.13 displays the message broadcasted by the robots communication transmitter along simulations of a duration of 4000 time instants. This message, bounded in [0, 1], is thereafter received by other robots in the surroundings, as explained in Sec. 3.4. In order to provide statistically relevant results, the message time series are constructed using 50 independent

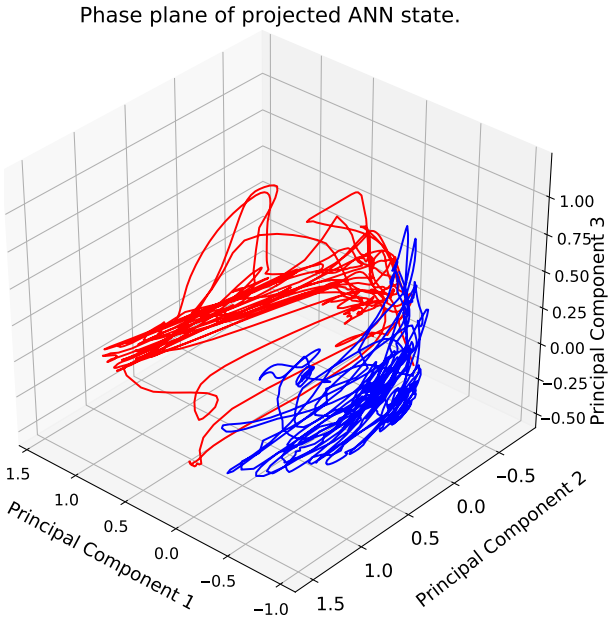


(a) Task Order: A2 ⟶ B2 *(b) Task Order: B2 ⟶ A2*

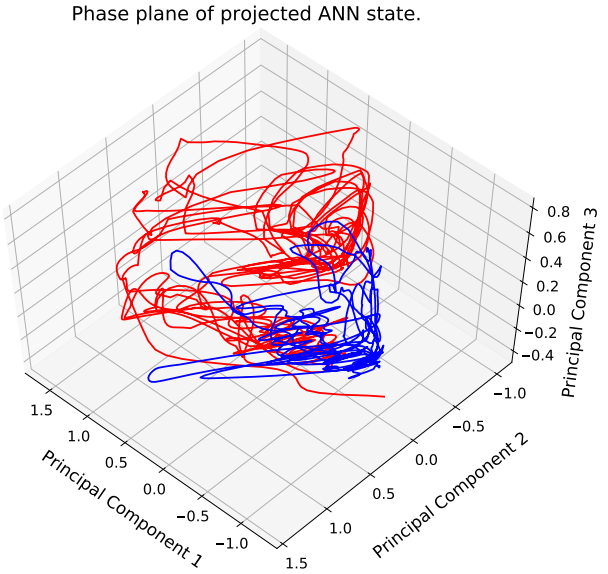
Figure 5.13: Message content transmitted by the robots that know the correct task (blue curves) and that are unable to acknowledge the current task (red curve) to be solved, within a simulation of 4000 time steps. The curves are built from statistics using a sample of 50 simulations. The dark curves illustrate the evolution of the median value of the messages and the shadow contours depict the first and third quantiles.

episodes, showing median values and quantiles. In Fig. 5.13a, the task order is {A2, B2}, while in Fig. 5.13b the sequence is {B2, A2}. The results of the figure exhibit that the communication is unidirectional, meaning that those robots that have access to the correct task notify it to the rest of the group (blue curve). On the contrary, the robots that cannot acquire the task information are silent by transmitting a message near to zero (red curve). The semantics underlying the message are clearly shown in the blue curves, in which Task A2 is represented by a zero and Task B2 is notified as a high level message (message content near to one). Even though the communication semantics and mechanics may seem remarkably simple and intuitive, the emergence of such an straightforward signalling communication, which is the main aim of this experiment, is utterly difficult in practice.

Finally, the state space trajectory of the CTRNN dynamics in Experiment 2 is studied. Just as in Experiment 1, the vector $\mathbf{u}(t)$, composed by the CTRNN neurons' firing rates, is projected onto a 3D space using PCA. Fig. 5.14 shows these trajectories in two scenarios: (a) using a robot that can perceive the current task to be solved and (b) representing the trajectory of the CTRNN of a robot unable to know the task to be executed. In both cases, the corresponding simulation is run for a total of 4000 time steps. Moreover, the first 2000 correspond to Task A2 and in time instants between 2000 and 4000 the robot has to accomplish Task B2. The blue curves correspond to the former time window and the red curve depicts the trajectory within the latter temporal interval. In both situations the red and blue dynamics are clearly separable, meaning that the CTRNN drastically modifies its dynamics in order to perform multiple tasks. This separability is more noticeable and evident in Fig. 5.14a because of the active signalling communication performed by the depicted robot.



(a) Trajectory of a robot's CTRNN dynamics that has access to the correct task.



(b) Trajectory of a robot's CTRNN dynamics that cannot know the correct task.

Figure 5.14: CTRNN state space trajectory of Experiment 2. It shows the trajectory of the neurons' firing rates projected onto 3 dimensions using PCA. The considered trajectory lasts 4000 simulation cycles and the blue curves correspond to the time instants of Task A2 while the red curve depicts the time instants when Task B2 is addressed.

Chapter 6

Conclusions and Future Research Lines

6.1 Conclusions

In this Master Thesis, the topics of task switching and communication emergence have been addressed using collective robotics systems. Continuous-time recurrent neural networks have been optimized using the evolutionary computation algorithm NEAT and the generalized ABCD Hebbian learning rules. Two experiments are proposed and solved. The first one is entirely devoted the task switching problem, so that robots have to simultaneously learn how to solve two different tasks and how to switch between them at the correct timing. The precise tasks are the pursuit of a mobile light source and the transportation of scattered small cubes. We showed that NEAT evolution can tune the CTRNN parameters and the network topology in order to proficiently solve the requested experiments. Specifically, after evolution, the robots are fully capable of solving both tasks. When required, they follow very closely the red light source while ignoring the cubic objects. In contrast, when the correct task to be tackled is the cube transportation, the robots forget about the mobile light and successfully transport the objects to the nest area. The task switching is carried out suitably with just a transit period approximately between 300 and 500 time steps. The results are supported by figures clearly showing the robot's behavior and using an statistically relevant sample of 50 simulations.

The second experiment is dedicated to the study of communication emergence when combined with task switching. Only one of the robots knows which of the two tasks have to be accomplished at each time instant. Therefore, this robot has to notify this information to the rest of the group. The communication semantics are not designed by us, they have to emerge from evolution. In this case, owing to the fact that the experiment is utterly more complex, both tasks consist of following a mobile light source. Nonetheless each light source has a different color and they are placed at opposite sides of the arena. We showed that communication can emerge from artificial evolution in the proposed experiment. The arised semantics are based on a unidirectional signalling process encoding the task to be addressed. The receiver robots are totally capable of reacting to the broadcasted message and accomplish the correct task. Although the objectives of the experiment were completely achieved, it should be mentioned that there are several limitations in the communication. For instance, it cannot scale properly if the number of robots is very large.

Finally, an equally important contribution of this Master Thesis was the improvement of a previously existing evolutionary swarm robotics simulator developed by us. The main upgrades are the addition of 3D graphics and renderings, realistic physics simulations and 3D

collision detections. Moreover, the algorithms used in the experiments, which are NEAT and Hebbian learning, have been implemented from zero for this Master Thesis.

6.2 Future research lines

Multiple futures lines of research have arised during the development of this Master Thesis. Firstly, our aim is that the software simulator remains in constant growth and improvement with the addition of new features and upgrades. For instance, we plan to include reinforcement learning algorithms to the simulator in order to complement the already existing artificial evolution implementation. Consequently, the robotics simulator can be leveraged under the frame of other future research works and projects or serve educational purposes.

Apart from the simulator prospects, the future research lines related to the collective robotics, evolutionary computation and ANNs, that we consider worth mentioning, are the following. One of the clearest expansions of this work is the study of multitasking and task switching using more than two tasks. Moreover, it would be highly interesting to design much more complex tasks than those presented. In this way, the limits of multitasking and artificial evolution can be assessed. Additionally, an utterly relevant study would be to investigate the task switching behaviors of robots when the correct task to be solved is not explicitly provided to any robot. This means that, instead of directly feeding the CTRNN with the task categorically encoded, the robot would have to explore and interact with the environment in order to discover the most rewarded actions. For instance, the CTRNNs of the robots can be aware of the current reward signal, which would be greater than zero if they are coping with the correct task. A third step in this research line would be that robots cannot acquire neither the encoded task nor the corresponding reward signal, and it is the Hebbian learning process, modulated by the reward, the one that is responsible of controlling the robot's behaviors.

In contrast to the task switching problem, there are several research ideas under the frame of the communication. The most immediate one is to increase the swarm sizes, both during evolution and evaluation, in order to reach the emergence of more robust and complex communication procedures. Furthermore, an interesting research line is to design and implement, under the frame of task switching, individual tasks that require communication by their own. Moreover, provided that the communication semantics required in each individual task are completely different, the robots not only would have to learn how to switch among tasks but also how to fully swap the communication semantics at runtime. Lastly, inspired by research works such as [105], a long term extension could be to explore the emergence more complex communication semantics and language beyond signalling.

Bibliography

[1] Ali Dorri, Salil S. Kanhere, and Raja Jurdak. Multi-agent systems: A survey. *IEEE Access*, 6:28573–28593, 2018.

[2] Wei Du and Wei Du. A survey on multi-agent deep reinforcement learning: from the perspective of challenges and applications. *Artificial Intelligence Review*, 54(5):3215–3238, 2021.

[3] J. Arokia Renjit and K. L. Shunmuganathan. Multi-agent-based anomaly intrusion detection. *Information Security Journal: A Global Perspective*, 20(4-5):185–193, 2011.

[4] Rutger Claes, Tom Holvoet, and Danny Weyns. A decentralized approach for anticipatory vehicle routing using delegate multiagent systems. *IEEE Transactions on Intelligent Transportation Systems*, 12(2):364–373, 2011.

[5] Bowen Baker, Ingmar Kanitscheider, Todor M. Markov, Yi Wu, Glenn Powell, Bob McGrew, and Igor Mordatch. Emergent tool use from multi-agent autocurricula. *CoRR*, abs/1909.07528, 2019.

[6] C. Ronald Kube and Hong Zhang. Collective robotics: From social insects to robots. *Adaptive Behavior*, 2(2):189–218, 1993.

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

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

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

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

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

[18] Maximilian Hüttenrauch, Adrian Sosic, and Gerhard Neumann. Guided deep reinforcement learning for swarm systems. *CoRR*, abs/1709.06011, 2017.

[19] Toshiyuki Yasuda and Kazuhiro Ohkura. Collective behavior acquisition of real robotic swarms using deep reinforcement learning. In *2018 Second IEEE International Conference on Robotic Computing (IRC)*, pages 179–180, 2018.

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

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] Rafael Sendra-Arranz and Álvaro Gutiérrez. Evolution of situated and abstract communication in leader selection and borderline identification swarm robotics problems. *Applied Sciences*, 11(8), 2021.

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

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

[28] 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.

[29] 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.

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

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

[32] 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.

[33] 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.

[34] 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.

[35] 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.

[36] 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.

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

[38] 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.

[39] 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.

[40] 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.

[41] 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.

[42] 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.

[43] 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.

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

[45] 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.

[46] 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.

[47] 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.

[48] Roderich Gross and Marco Dorigo. Towards group transport by swarms of robots. *International Journal of Bio-Inspired Computation*, 1(1-2):1–13, 2009.

[49] Jianing Chen, Melvin Gauci, Wei Li, Andreas Kolling, and Roderich Gross. Occlusion-based cooperative transport with a swarm of miniature mobile robots. *IEEE Transactions on Robotics*, 31(2):307–321, 2015.

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

[51] Michael J.B. Krieger and Jean-Bernard Billeter. The call of duty: Self-organised task allocation in a population of up to twelve mobile robots. *Robotics and Autonomous Systems*, 30(1):65–84, 2000.

[52] Wenguo Liu, Alan F. T. Winfield, Jin Sa, Jie Chen, and Lihua Dou. Towards energy optimization: Emergent task allocation in a swarm of foraging robots. *Adaptive Behavior*, 15(3):289–305, 2007.

[53] C. Jones and M. Mataric. Adaptive division of labor in large-scale minimalist multi-robot systems. *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No.03CH37453)*, 2:1969–1974 vol.2, 2003.

[54] Genci Capi. Robot task switching in complex environments. In *2007 IEEE/ASME international conference on advanced intelligent mechatronics*, pages 1–6, 2007.

[55] Genci Capi, Genci Pojani, and Shin-Ichiro Kaneko. Evolution of task switching behaviors in real mobile robots. In *2008 3rd International Conference on Innovative Computing Information and Control*, pages 495–495, 2008.

[56] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. *CoRR*, abs/1703.03400, 2017.

[57] 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.

[58] 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.

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

[60] 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.

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

[62] 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.

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

[64] 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.

[65] 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.

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

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

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

[69] 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.

[70] 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.

[71] 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.

[72] Rafael Sendra Arranz. Design and development of continuous-time recurrent neural networks evolution for cooperative distributed multi-agent systems. Master's thesis, Escuela Técnica Superior de Ingenieros de Telecomunicación. Universidad Politécnica de Madrid, Madrid, Spain, 2021.

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

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

[75] 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.

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

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

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

[79] 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.

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

[81] 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.

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

[83] 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.

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

[85] 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.

[86] Risto Miikkulainen, Jason Zhi Liang, Elliot Meyerson, Aditya Rawal, Daniel Fink, Olivier Francon, Bala Raju, Hormoz Shahrzad, Arshak Navruzyan, Nigel Duffy, and Babak Hodjat. Evolving deep neural networks. *CoRR*, abs/1703.00548, 2017.

[87] Adam Gaier and David Ha. Weight agnostic neural networks. *CoRR*, abs/1906.04358, 2019.

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

[89] J.D. Schaffer, D. Whitley, and L.J. Eshelman. Combinations of genetic algorithms and neural networks: a survey of the state of the art. In *[Proceedings] COGANN-92: International Workshop on Combinations of Genetic Algorithms and Neural Networks*, pages 1–37, 1992.

[90] Samir W. Mahfoud. *Niching Methods for Genetic Algorithms*. PhD thesis, USA, 1995. UMI Order No. GAX95-43663.

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

[92] R.M. Palnitkar and J. Cannady. A review of adaptive neural networks. In *IEEE SoutheastCon, 2004. Proceedings.*, pages 38–47, 2004.

[93] D. Hebb. The organization of behavior. *New York: Wiley*, 1949.

[94] E. Najarro and S. Risi. Meta-learning through hebbian plasticity in random networks. *ArXiv*, abs/2007.02686, 2020.

[95] Sebastian Risi, Charles E Hughes, and Kenneth O Stanley. Evolving plastic neural networks with novelty search. *Adaptive Behavior*, 18(6):470–491, 2010.

[96] Geoffrey E. Hinton and S. Nowlan. How learning can guide evolution. *Complex Syst.*, 1, 1987.

[97] Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. `http://pybullet.org`, 2016–2021.

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

[99] Nikolaus Hansen. The CMA evolution strategy: A tutorial. *CoRR*, abs/1604.00772, 2016.

[100] 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.

[101] 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.

[102] Karl Pearson F.R.S. Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572, 1901.

[103] Ian T. Jolliffe and Jorge Cadima. *Principal component analysis: a review and recent developments*, volume 374. 2016.

[104] Kyriacos Nikiforou. *The dynamics of continuous-time recurrent neural networks and their relevance to episodic memory.* PhD thesis, Imperial College of Science, Technology and Medicine, 2019.

[105] Nicolas Cambier, Roman Miletitch, Vincent Frémont, Marco Dorigo, Eliseo Ferrante, and Vito Trianni. Language evolution in swarm robotics: A perspective. *Frontiers in Robotics and AI*, 7:12, 2020.

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

[107] 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.

[108] 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.

[109] 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_task_switching_NEAT",
        "topology" : {
            "dt" : 0.1,
            "time_scale" : 1,
            "stimuli": {
                "I1" : {"n" : 4, "sensor" : "yellow_light_sensor"},
                "I2" : {"n" : 4, "sensor" : "red_light_sensor"},
                "I3" : {"n" : 4, "sensor" : "distance_sensor3D"},
                "I4" : {"n" : 4, "sensor" : "color_sensor"},
                "I5" : {"n" : 1, "sensor" : "ground_sensor"},
                "I6" : {"n" : 1, "sensor" : "object_grasped_sensor"},
                "I7" : {"n" : 1, "sensor" : "task_sensor"}
            },
            "neuron_model" : "rate_model",
            "synapse_model" : "static_synapse",
            "ensembles": {
                "H" : {"n" : 5, "params" : {}},
                "OUT_MOT" : {"n" : 2, "params" : {"activation" : "tanh"}},
                "OUT_GRASP" : {"n" : 3, "params" : {}}
            },
            "outputs" : {
                "outA" : {"ensemble" : "OUT_MOT", "actuator" : "joint_velocity_actuator",
                    ↪ "enc": "real"},
                "outD" : {"ensemble" : "OUT_GRASP", "actuator" : "grasp_actuator", "enc":
                    ↪ "real"}
            },
            "synapses" :  {
                "I1-H" : {"pre":"I1","post":"H", "trainable":true, "p":1.0},
                "I2-H" : {"pre":"I2","post":"H", "trainable":true, "p":1.0},
                "I3-H" : {"pre":"I3","post":"H", "trainable":true, "p":1.0},
                "I4-H" : {"pre":"I4","post":"H", "trainable":true, "p":1.0},
                "I5-H" : {"pre":"I5","post":"H", "trainable":true, "p":1.0},
                "I6-H" : {"pre":"I6","post":"H", "trainable":true, "p":1.0},
                "I7-H" : {"pre":"I7","post":"H", "trainable":true, "p":1.0},
                "H-MOT" : {"pre":"H","post":"OUT_MOT", "trainable":true, "p":1.0},
                "H-GRASP" : {"pre":"H","post":"OUT_GRASP", "trainable":true, "p":1.0}
            },
            "decoding" : {
                "outA" : {"scheme" : "IdentityDecoding", "params" : {"is_cat" : false}},
                "outD" : {"scheme" : "SoftmaxDecoding", "params" : {"is_cat" : true}}
            },
            "learning_rule" : {
                "rule" : "generalized_hebbian",
                "reward" : null
            }
        },
        "algorithm" : {
            "name" : "NEAT",
            "evolvable_object" : "robotA",
            "population_size" : 300,
            "generations" : 2000,
            "evaluation_steps" : 4000,
            "num_evaluations" : 8,
            "fitness_function" : "task_switching",
            "populations" : {
                "p1" : {
```

```
                    "objects" : ["synapses:weights:all", "neurons:bias:all",
                        ↪ "neurons:tau:all", "learning_rule:params:all"],
                    "max_vals" : [4,  3, 1.2, 2],
                    "min_vals" : [-4, -3, -0.9, -2],
                    "params" : {
                        "encoding" : "real",
                        "p_weight_mut" : 0.1,
                        "p_node_mut" : 0.02,
                        "p_conn_mut" : 0.15,
                        "compatib_thresh" : 0.35,
                        "c1" : 1,
                        "c2" : 1,
                        "c3" : 0.6,
                        "species_elites" : 2
                    }
                }
            }
        },
        "world":{
            "engine" : "3D",
            "world_delay" : 1,
            "height" : 10,
            "width" : 10,
            "objects" : {
                "robotA" : {
                    "type" : "robot",
                    "num_instances" : 3,
                    "controller" : "neural_controller",
                    "sensors" : {
                        "color_sensor" : {"n_sectors" : 4, "range" : 3},
                        "yellow_light_sensor" : {"n_sectors" : 4, "range" : 20},
                        "red_light_sensor" : {"n_sectors" : 4, "range" : 20},
                        "IR_receiver" : {"n_sectors" : 4, "range" : 4, "msg_length" : 2,
                            ↪ "selection_scheme" : "random"},
                        "distance_sensor3D" : {"n_sectors" : 4, "range" : 3},
                        "object_grasped_sensor" : {},
                        "ground_sensor" : {},
                        "task_sensor" : {}
                    },
                    "actuators" : {
                        "joint_velocity_actuator" : {"joint_ids" : [0, 1], "max_velocity"
                            ↪ : 15},
                        "wireless_transmitter" : {"quantize": true, "range" : 4,
                            ↪ "msg_length":2},
                        "grasp_actuator" : {}
                    },
                    "initializers" : {
                        "positions" : {"name" : "random_uniform",  "params" : {"low" : [
                            ↪ -1, -3], "high" : [1, -2], "size" : 2}},
                        "orientations" : {"name" : "random_uniform",  "params" : {
                            ↪ "low":0, "high" : 6.28, "size" : 1}}
                    },
                    "perturbations" : {
                    },
                    "params" : {"trainable" : true}
                },
                "light_yellow" : {
                    "type" : "light_source",
                    "num_instances": 1,
                    "initializers" : {
                        "positions" : {"name" : "fixed_random", "params" : {
                            ↪ "possible_values": [[0, 3.5], [3.5, 0], [-3.5, 0]]}}
                    },
                    "params" : {"range" : 20, "color" : "yellow"}
                },
                "ground_area_grey" : {
                    "type" : "ground_area",
                    "num_instances": 3,
                    "initializers" : {
                        "positions" : {"name" : "fixed", "params" : {"fixed_values": [[0,
                            ↪ 3.5], [3.5, 0], [-3.5, 0]]}}
                    },
                    "params" : {"radius" : 1.5, "color" : "grey"}
                },
                "cube_blue" : {
                    "type" : "cube",
                    "num_instances": 2,
                    "initializers" : {
                        "positions" : {"name" : "random_uniform",  "params" : {"low" : [
                            ↪ -1, -1], "high" : [1, 2], "size" : 2}}
                    },
```

```
                    "params" : {"mass" : 0.1, "side_len" : 0.2, "color" : "blue"}
                },
                "light_red" : {
                    "type" : "light_source",
                    "num_instances": 1,
                    "controller" : "light_orbit_controller",
                    "initializers" : {
                        "positions" : {"name" : "random_uniform", "params" : {"low":-1,
                            ↪ "high" : 1, "size" : 2}}
                    },
                    "params" : {"range" : 20, "color" : "red"}
                },
                "task_scheduler" : {
                    "type" : "task_scheduler",
                    "num_instances": 1,
                    "params" : {"num_tasks" : 2, "num_slots" : 2, "total_timesteps" : 4000
                        ↪ }
                }
            }
        }
    }
```

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. An important remark to be highlighted is that it is posed under a theoretical perspective and the experiments are not tested in real robots. Even though it is difficult to highlight direct social, ethical or environmental impacts, the use of the systems, ideas, concepts or methodologies of this Master Thesis can be undoubtedly impactful. Aside from the impacts described in the next section, an equally important effect of this Master Thesis is the educational and research contribution of our work. Specifically, the evolutionary robotics simulator can be leveraged in order to develop further research works and also serve educational purposes. Furthermore, the results and setups of this Master Thesis, in relation to the task switching and communication in collective robotics, can contribute to the development of these research lines.

B.2 Description of the relevant project related problems

B.2.1 Social impact

Collective and swarm robotics systems have been applied to several problems with a direct and highly relevant social impact. The most noticeable example is the application in tasks of seek and rescue (see e.g [106] and [107]) of victims of a disaster. It should be pointed out that in these problems, an efficient communication, is of crucial importance. For instance, one of the robots might have discovered a victim of, say, an earthquake, and it has to notify it to the rest of the robots in order to cooperatively act in consequence. Multitasking is also very relevant because robots could have to assume multiple tasks or roles within the rescue.

B.2.2 Economical impact

An important characteristic of collective robotics is that its robots are remarkably simple and self-sufficient. This fact can be readily economically impactful by replacing other traditional labors that use much more expensive machinery and processes to be completed. In general terms, collective robotics systems, such as the one described in this document, can automatize existing tasks, reducing operative costs.

B.2.3 Environmental impact

The use of many robots that interact in order to deal with a common problem is extremely relevant in environmental problems. These concerns commonly involve the execution of prevention or contingency actions over a vast geographical area. Therefore, collective robotics systems can have a noticeable environmental impact as they can cooperate in order to deal with the problem much more efficiently and concertedly. Some clear use case examples are the fire prevention or contingency in forests (see e.g. [108]) or the garbage collection using swarm of robots (see e.g. [109]). In the latter case, it can be also noticed that the application has straight similarilies with the cube transportation task faced in this Master Thesis.

B.2.4 Ethical impact

In relation the ethical impact of this Master Thesis, it has to be mentioned that it fulfills all the ethical guidelines for designing and implementing trustworthy artificial intelligence as established in the EU [1].

B.3 Conclusions

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

[1] https://digital-strategy.ec.europa.eu/en/library/ethics-guidelines-trustworthy-ai

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**
		480	30€	**14.400 €**
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				**14.400 €**
Material costs				**500 €**
General expenses (Indirect cost)	15% of direct cost			**2.235 €**
Industrial benefits	6% of direct and indirect costs			**1.028,1 €**
IVA (21%)				**3.814,251 €**
Total				**21.977,351 €**

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