Automatic Design of Controllers for Robotic Swarms

(ロボティックスワームのための制御器の自動的設計)

Motoaki Hiraga
(平賀 元彰)

Department of Mechanical Systems Engineering
Graduate School of Engineering
Hiroshima University

March 2022

Abstract

This thesis presents automatic design methods for designing controllers for robotic swarms. One of the challenges in the field of swarm robotics is designing control software for a robotic swarm. The most common approach in swarm robotics follows a trial and error process to design a controller. This method is effective if the desired collective behavior is simple enough for the designer to understand and program into the controller. However, this method is guided only by the designer's intuition and experience. As an alternative method, the automatic design method develops controllers by transforming the design problem into an optimization problem. This thesis focuses on the evolutionary robotics approach, which is the most often used automatic design method. The evolutionary robotics approach utilizes an evolutionary algorithm to optimize the parameters of the controller. The traditional evolutionary robotics approach uses artificial neural networks as robot controllers. Typically, the structure of the neural network is determined by the designer, and the synaptic weight values of the neural network are optimized by an evolutionary algorithm. So far, there has been little progress in studies using automatic design methods within the swarm robotics community. This thesis contributes to the swarm robotics community from the following two aspects.

First, this thesis presents how the evolutionary robotics approach could be applied to develop controllers for robotics swarms to perform tasks that are difficult to design controllers by hand. A common approach based on a trial and error process to design a robot controller is guided by the designer. Therefore, a robotic swarm would not exhibit collective behavior beyond the designer's intention. This method would not work well when the task is too complex for the designer to design the controller by hand. The evolutionary robotics approach has the potential to produce collective behavior that might be difficult to develop manually by the designer. Moreover, collective behavior developed by the evolutionary robotics approach could give insights into how robotic swarms should be composed to address complex tasks.

Second, this thesis presents novel evolutionary robotics approaches to design controllers for robotic swarms. The traditional evolutionary robotics approach uses relatively simple artificial neural networks as controllers. This thesis aims to develop an evolutionary robotics approach using neural networks with a special structure frequently used in the framework of reservoir computing. The proposed approach could be an alternative to the traditional neural networks that could accelerate the evolutionary process even with complex structured neural networks. In addition, this thesis proposes an approach to optimizes both the synaptic weight values and the topological structure of the neural network. In the traditional evolutionary robotics approach, only the synaptic weight values of the neural network are optimized by an evolutionary algorithm. This approach might restrict the behavior of robots or might have unsuitable structures within the controller. The proposed method could overcome these limitations in the traditional evolutionary robotics approach.

Acknowledgements

There are many people I would like to thank for their contributions to this thesis. This work would not have been possible without their supports.

First and foremost, I would like to thank my supervisor, Prof. Kazuhiro Ohkura, for helpful advice on research and for providing all kinds of supports during my academic life at Hiroshima University. Also, I would like to thank the Ph.D. thesis committee members, Prof. Nobutaka Wada, Prof. Soichi Ibaraki, and Prof. Yoshiyuki Matsumura, for revising the thesis and providing insightful comments.

I would like to thank Prof. Yoshiyuki Matsumura, Assoc. Prof. Toshiyuki Yasuda, and Assoc. Prof. Masanori Goka for fruitful discussion and helpful advice on the research. I am especially grateful to Yasuda-sensei for his supports during my bachelor's course and partly in my master's course at Hiroshima University.

I would thank past and current members of Machine Intelligence and Systems A Laboratory (formerly Manufacturing Systems A Laboratory) for many help and supports. I enjoyed my wonderful time at Hiroshima University due to my friends and colleagues.

This work was partially supported by JSPS KAKENHI Grant Number JP21J14922.

At last, I would like to thank my family for many supports. Without their supports, it would be impossible for me to complete my studies.

January 2022
Motoaki Hiraga

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Recent advances in artificial intelligence and robotics technologies have been attracting considerable interest. These technologies have been or will be applied to various real-world applications, such as self-driving cars, industrial robots, smart agriculture, rescue robots, or medical robots. In addition, many countries are attempting to shape a future society that incorporates artificial intelligence and robotics technologies. For example, Japan is aiming for Society 5.0 [13, 41] with the help of artificial intelligence and robotics technologies. In the future society, more robots and artificial intelligence systems will be ubiquitous in our everyday lives.

One of the challenges in shaping future robot systems could be how to design groups of robots that coordinate and cooperatively solve or perform a task. *Swarm robotics* [22, 25, 48, 96] is a promising research field that focuses on this challenge. It is inspired by the collective behavior of animals, such as flocks of birds, schools of fish, and colonies of ants, which is a widespread phenomenon often observed in biological systems [14, 19, 110, 118]. Examples of collective behavior in biological swarms are shown in Fig. 1.1. Starlings are known to form a flock that twist, turn, and swirl across the sky while diluting the risk of attack by predators. Schools of fish swim together by turning and twisting synchronously, which confuses predators and reduces the risk of being attacked. Ants cooperate to transport objects that are too heavy to carry alone. These swarm systems are composed of large groups of individuals and exhibit complex collective behavior. In addition, these collective behavior emerge from local interactions among individuals without relying on a global plan or centralized leader. The study of intelligent systems, both natural and artificial, inspired by the collective behavior of biological swarms is called *swarm intelligence* [5, 9, 21]. Swarm intelligence is also known as a subfield of artificial intelligence.

The field of swarm robotics has emerged as the application of swarm intelligence to robot systems. Swarm robotics focuses on the coordination of a large group of autonomous robots, with emphasis on the physical embodiment of robots. Therefore, swarm robotics could

(a) Flock of starlings (by Airwolfhound, licensed under CC BY-SA 2.0).

(b) School of fish (by Sam Howzit, licensed under CC BY 2.0).

(c) Cooperative transport in a group of ants (by Axel Rouvin, licensed under CC BY 2.0).

(d) Swarm of bees (by Harlequeen, licensed under CC BY 2.0).

Fig. 1.1. Examples of collective behavior in biological swarms.

be defined as *embodied swarm intelligence*. Similar to biological swarms, robotic swarms operate without relying on a centralized controller. Typically, a robotic swarm is composed of homogeneous robots that are incapable or inefficient at performing tasks alone. However, by utilizing swarm intelligence principles, robotic swarms perform tasks beyond the capability of a single robot. Robotic swarms are expected to have the following three system-level properties; (i) fault tolerance for operating despite failures in the individuals or disturbances in the environment, (ii) flexibility for generating modularized solutions to the different tasks, and (iii) scalability for operating under a wide range of group sizes.

Designing a controller for a robotic swarm is a challenging problem. In general, the designer needs to define the controller and behavior at the level of individual robots; however, the goal is to obtain the desired collective behavior that emerges from individual-level behaviors and interactions among robots. Therefore, the designer has to go through a two-stage process to design a controller; i.e., (i) defining the behavior of individual robots that the designer believes to be required to produce the desired collective behavior and (ii) designing the

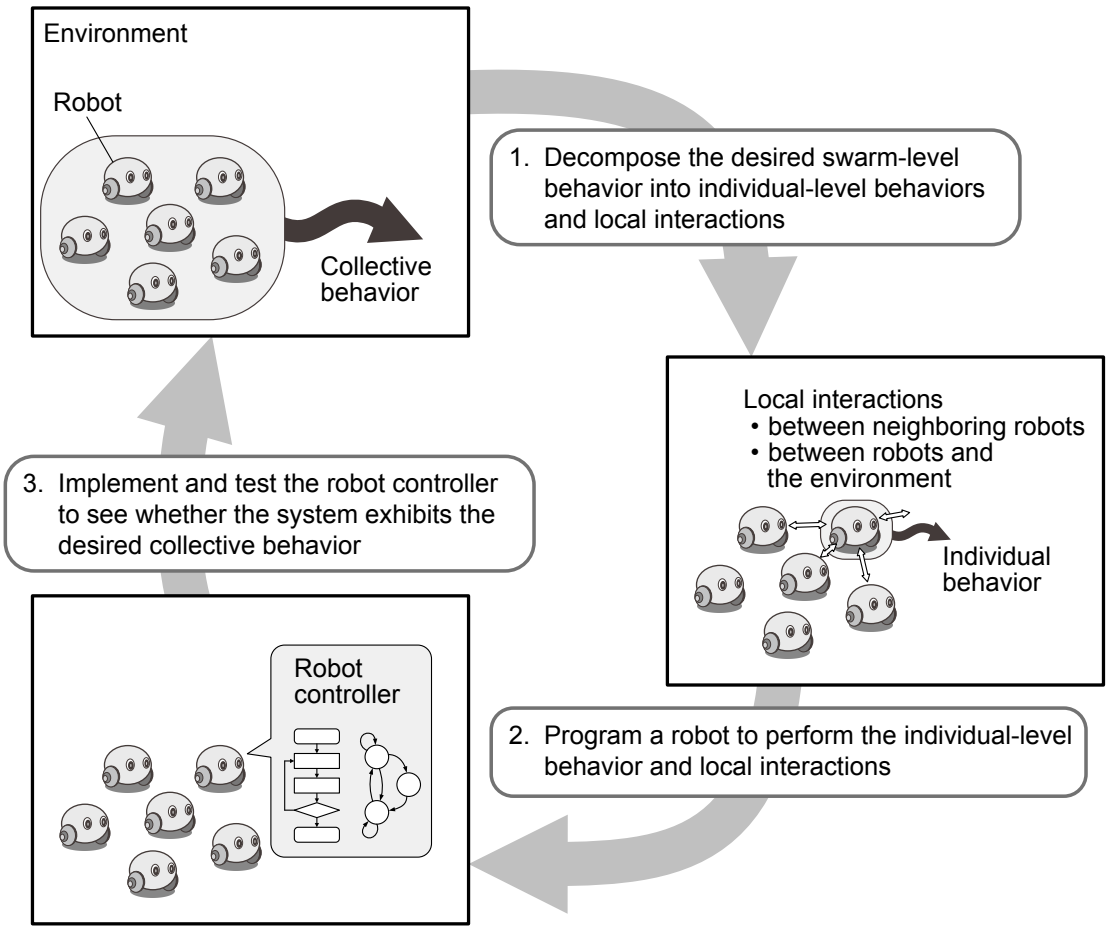


Fig. 1.2. Process flow of a behavior-based design method.

controller at the level of individual robots. This difficulty in designing a controller for a robotic swarm is referred to as the *design problem* [113].

Design methods for controllers could be classified into two categories, i.e., *behavior-based design* and *automatic design* methods [10]. The behavior-based design method is the most common approach in the field of swarm robotics. A brief overview of a behavior-based design method is illustrated in Fig. 1.2. This method follows a trial and error process in which the individual behavior of each robot is implemented, tested, and improved until the robotic swarm emerges the desired collective behavior. This method could design a controller efficiently according to the task if the desired collective behavior is simple enough for the designer to understand and program into the controller. Various studies using the behavior-based design method have been reported (e.g., [77, 79, 95]). However, this method is guided only by the designer's intuition and experience for designing controllers to obtain the desired collective behavior.

As an alternative method, the automatic design method is an effective technique to reduce

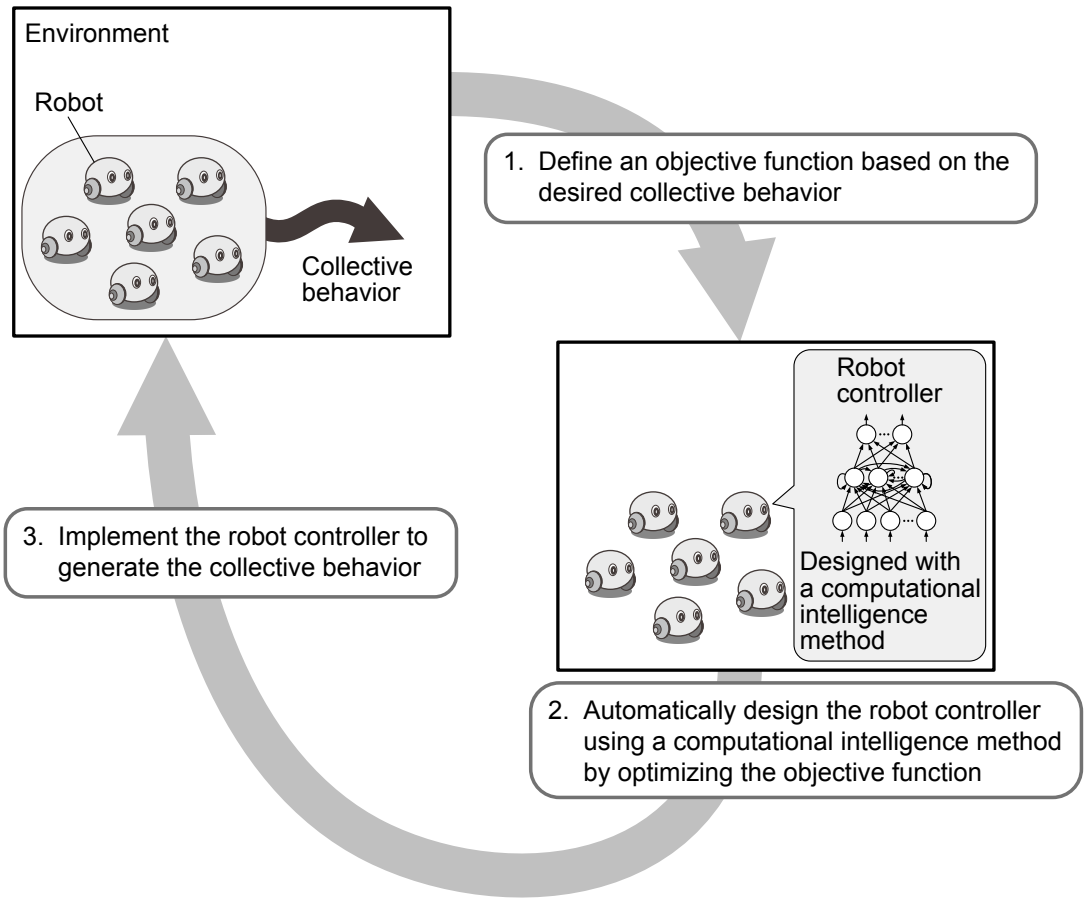


Fig. 1.3. Process flow of an automatic design method.

the effort of the designer required to develop a controller of a robotic swarm [39]. An overview of an automatic design method is shown in Fig. 1.3. In this method, the controller for the robotic swarm is generated automatically by transforming the design problem into an optimization problem. The widely used automatic design method in swarm robotics is the *evolutionary robotics* approach [90, 113]. The evolutionary robotics approach utilizes evolutionary computation to optimize the parameters of the controller. In the most common evolutionary robotics approach, artificial neural networks are applied to the robot controllers. Typically, the structure of the neural network is determined by the designer, and the synaptic weight values of the neural network are optimized.

1.1 Aim and Objectives

This thesis focuses on automatic design methods, specifically evolutionary robotics approaches, for designing controllers for robotic swarms. So far, there has been little progress in studies using automatic design methods within the swarm robotics community [25]. There-

fore, this thesis aims to contribute to the swarm robotics community from the following aspects.

The first goal of this thesis is to develop controllers for robotic swarms to perform tasks that are difficult to design controllers by hand. The collective behavior generated by the controller designed with a behavior-based design method is guided by the designer. Therefore, a robotic swarm would not exhibit collective behavior beyond the designer's intention. This method would not work well when the task is too complex for the designer to program the controller by hand. The evolutionary robotics approach has the potential to produce collective behavior that might be difficult to develop manually by the designer. Moreover, collective behavior developed by the evolutionary robotics approach could give insights into how robotic swarms should be composed to address complex tasks.

The second goal is to develop novel evolutionary robotics approaches for designing controllers for robotic swarms. This thesis aims to develop an evolutionary robotics approach using neural networks with a special structure often used in the framework of reservoir computing [58, 80]. Most studies using the evolutionary robotics approach apply relatively simple structured neural networks. The proposed approach could be an alternative to the traditional neural networks that could accelerate the evolutionary process even with complex structured neural networks. This thesis also proposes an approach to optimizes both the synaptic weight values and the topological structure of the neural network. In the traditional evolutionary robotics approach, only the synaptic weight values of the neural network are optimized by the evolutionary computation technique. This approach might restrict the behavior of robots or might have unsuitable structures within the controller. The proposed method could overcome these limitations in the traditional evolutionary robotics approach.

1.2 Structure of the Thesis

As described above, the first goal is to develop controllers for robotic swarms using an evolutionary robotics approach. Based on this topic, Chapters 3 and 4 present case studies using the evolutionary robotics approach. In addition, Chapter 5 gives further analysis on collective behavior developed in Chapter 4 to provide insights into how robotic swarms should be composed.

The proposal of novel evolutionary robotics approaches is described in Chapters 6 and 7. Chapter 6 proposes an evolutionary robotics approach that applies the echo state networks, i.e., neural networks often used in reservoir computing, as controllers for a robotic swarm. The evolutionary robotics approach that evolves both the weight values and the structure of neural networks is proposed in Chapter 7.

The overall structure of this thesis is illustrated in Fig. 1.4. In this thesis, all of the experiments are conducted in computer simulations. A summary of each chapter is described

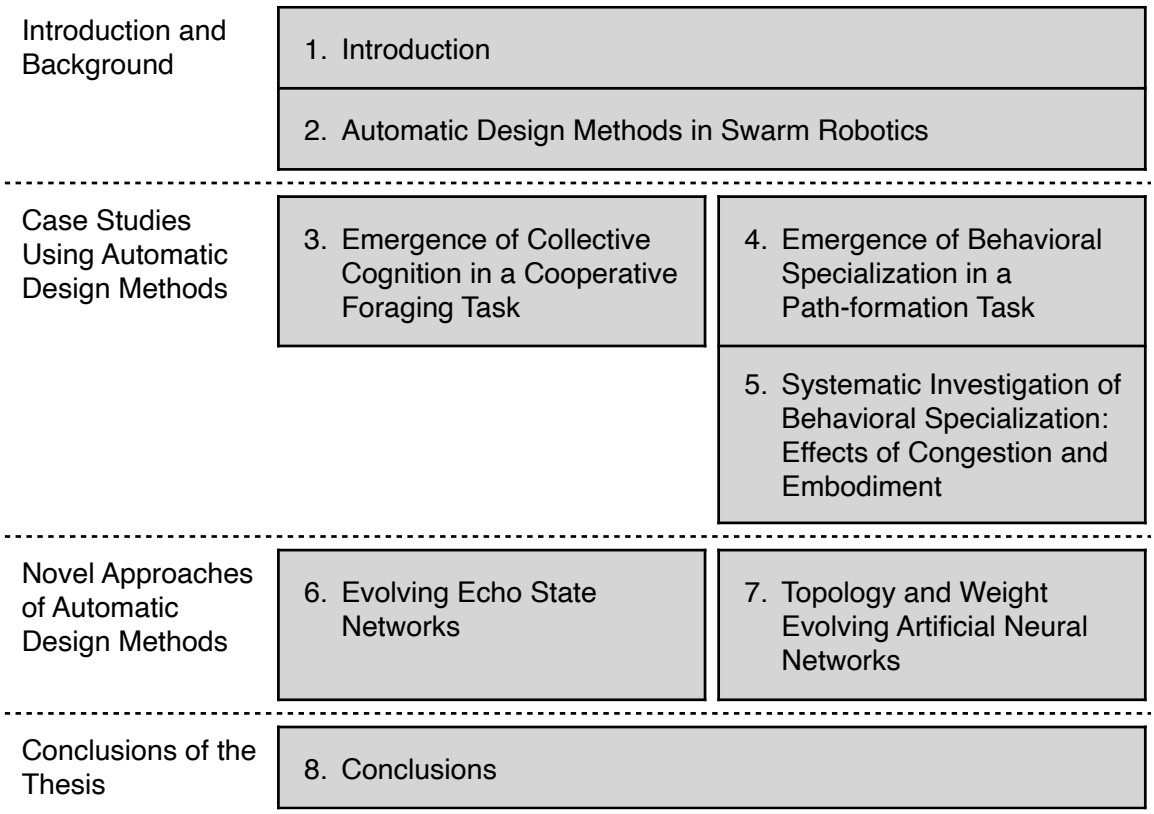


Fig. 1.4. Overview of the thesis structure.

as follows.

- Chapter 2 provides a brief review of automatic design methods in swarm robotics. In addition, this chapter briefly describes an introduction to evolutionary robotics, including an introduction to evolutionary computation and approaches to evolving neural networks.
- Chapter 3 focuses on the collective cognition by robotic swarms. In biological swarms, individual cognition is enhanced by communication and cooperation with other individuals. A single individual only has a limited cognition ability; however, it could be extended beyond its cognition ability when considered as a swarm. Benefitting from collective cognition, biological swarms can exhibit sophisticated collective behavior and decision-making processes. On the other hand, the robotic swarms have to rely on collective cognition more than biological swarms when considering the limitation in sensory capabilities and the cost of each robot. This chapter aims to develop controllers using the evolutionary robotics approach to address a foraging task that requires collective cognition. In this task, the robotic swarm has to distinguish between two types of objects and cooperatively transport one of them to the goal area. Additional experiments are conducted to examine the scalability and flexibility of the developed

controllers.

- Chapter 4 aims to develop controllers to address a path-formation task with congested situations. Typically, robotic swarms are conducted to have high redundancy in the number of robots. However, in situations where multiple robots gather in a spatially limited environment, robots tend to interfere with each other. This chapter develops a controller for a robotic swarm to exhibit behavioral specialization to manage congestion in a path-formation task.
- Chapter 5 further discusses the behavioral specialization developed in Chapter 4. More specifically, this chapter focuses on the effect of the embodiment of robots on collective behavior and specialization. The studies on swarm robotics emphasize the importance of the embodiment of robots. However, so far, only a few studies have discussed how the embodiment influences the collective behavior of robotic swarms. In this chapter, the experiments are conducted by varying the size of robots to change the degree of congestion. Additionally, the experiments are conducted with and without considering collisions among robots to discuss the effect of the robot embodiment. These experiments show that the embodiment of the robots is also an essential feature to discuss swarm performance and specialization.
- Chapter 6 proposes an evolutionary robotics approach that applies echo state networks as controllers for a robotic swarm. The main characteristics of the echo state network are that the hidden layer is generated with sparse and random connections, and only the weight values connected to the output layer are trained. The proposed approach utilizes evolutionary computation to optimize the output weight values of the echo state network. The performance of the proposed approach is compared with a typical method that uses a traditional neural network controller.
- Chapter 7 focuses on the evolutionary robotics approach that evolves both the synaptic weight values and the topological structure of the neural network controller. The algorithms that evolve both the values of the synaptic weights and the topological structure of the neural networks are called Topology and Weight Evolving Artificial Neural Networks (TWEANNs). This chapter applied the TWEANN algorithm called Mutation-Based Evolving Artificial Neural Network (MBEANN), which only employs mutations to evolve neural networks. For comparison, a widely used TWEANN algorithm called NeuroEvolution of Augmenting Topologies (NEAT) is employed to develop robot controllers. The evolved controllers are evaluated in a cooperative transportation task performed by a robotic swarm.
- Chapter 8 concludes the thesis and discusses future research directions.

Chapter 2

Automatic Design Methods in Swarm Robotics

One of the main challenges in swarm robotics focuses on designing controllers for robotic swarms [10]. Typically, the controllers are designed by a trial and error process until the robotic swarm has emerged an acceptable collective behavior to perform the task. However, this method is guided only by the designer's intuition and experience for designing controllers to obtain the desired collective behavior. As an alternative method, the automatic design method develops controllers by transforming the design problem into an optimization problem [39]. This method could reduce the effort required by the designer to develop controllers of a robotic swarm.

This chapter provides a brief introduction to automatic design methods in swarm robotics. The automatic design methods could be classified into two approaches, i.e., *evolutionary robotics* and *reinforcement learning*. First, this chapter briefly introduces the two approaches. Then, Section 2.1 presents an introduction to evolutionary robotics. A brief review of evolutionary robotics approaches in swarm robotics is described in Section 2.2. Finally, Section 2.3 concludes this chapter.

Evolutionary Robotics

The evolutionary robotics approach [90, 113] is a widely used automatic design method in swarm robotics. This approach uses an evolutionary algorithm [30] to develop and optimize controllers for a robotic swarm. Typically, this approach uses *evolving artificial neural networks* [122], also known as *neuroevolution* [33], to develop controllers that are represented by artificial neural networks. Further details on the evolutionary robotics approach are described in Section 2.1.

Reinforcement Learning

Reinforcement learning is a machine learning technique where a robot learns its actions by interacting with an environment [111]. The robot could obtain a numerical reward by trial and error interactions. The goal is to train the robot to select actions that maximize the expected cumulative reward. So far, studies that have applied reinforcement learning for designing a controller for a robotic swarm are rarely seen in the field of swarm robotics. However, few studies have applied reinforcement learning to multi-robot systems [92, 107].

In recent years, neural networks with multiple hidden layers, which are also called *deep neural networks*, have attracted considerable interest [45, 73]. Recent advances in deep neural networks have motivated the studies of reinforcement learning. The combination of deep neural networks and reinforcement learning is called *deep reinforcement learning* [85]. Few recent studies have attempted to use deep reinforcement learning or novel reinforcement learning algorithms to design a controller for a robotic swarm (e.g., [56, 60, 89, 121, 123]).

2.1 Evolutionary Robotics

This section provides a brief introduction to evolutionary robotics. The experiments on developing controllers for autonomous robots were first reported in the early 1990s by the University of Sussex [16, 52], the Swiss Federal Institute of Technology in Lausanne [34], and the University of Southern California [76]. These studies have triggered the field of *evolutionary robotics*. In addition, evolutionary robotics techniques for designing swarm robotics systems are often referred to as *evolutionary swarm robotics* [113]. Typically, the evolutionary robotics approach uses an evolutionary algorithm to develop and optimize controllers. In a traditional evolutionary robotics approach, controllers are represented by artificial neural networks. The rest of this section describes an introduction to evolutionary computation, neuroevolution, and a typical evolutionary robotics approach for designing controllers for a robot.

2.1.1 Evolutionary Computation

Evolutionary computation is a research field within computational intelligence that studies optimization methods inspired by biological evolution [30, 31]. The main inspiration comes from the Darwinian principle [18] of natural selection and survival of the fittest. In general, population-based optimization algorithms[*1] inspired by biological evolution are called *evolutionary algorithms*.

During the 1960s, three pioneering algorithms were developed separately, i.e., genetic

[*1] In contrast to single-point optimization algorithms such as hill-climbing algorithms and simulated annealing algorithms, a population-based optimization algorithm consists of a set of points (or candidate solutions) that cooperatively search for the optimum solution.

Algorithm 2.1: Pseudo-code of a typical evolutionary algorithm.

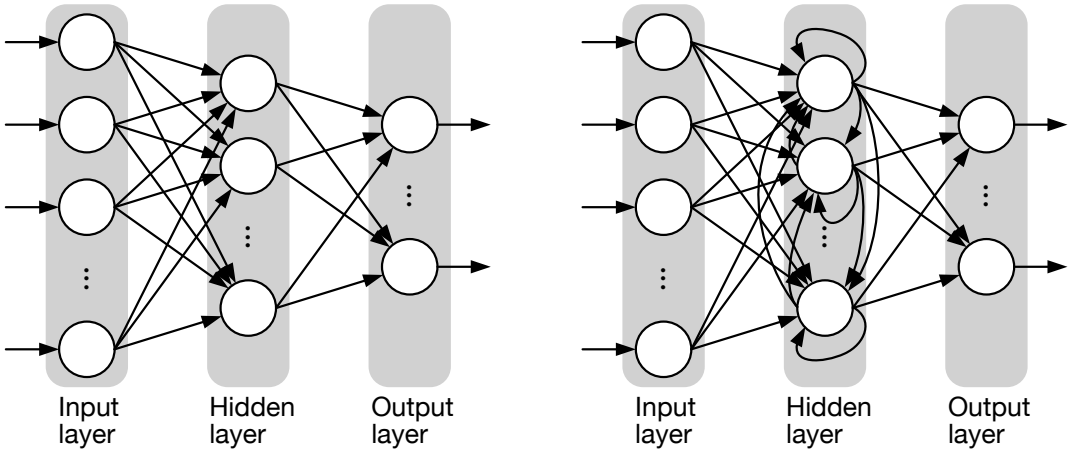
```
Initialization: Generate an initial population of individuals;
Evaluation: Calculate the fitness value of each individual;
repeat
    Selection: Select parents from the current population;
    Recombination: Generate offspring by recombining parents;
    Mutation: Mutate each offspring;
    Evaluation: Calculate the fitness value of each offspring;
    Selection: Select individuals to form the new population for the next generation;
until Terminal condition;
```

algorithms [43, 55], evolution strategies [7, 99, 100], and evolutionary programming [35–37]. Subsequently, the fourth mainstream algorithm called genetic programming [3, 68, 69] has emerged around 1990. In 1993, the term *evolutionary computation* was coined to put these studies together. Other than the four fundamental algorithms, differential evolution [94, 108] was proposed as a simple and effective algorithm to optimize real-valued functions. Additionally, population-based optimization algorithms inspired by the collective behavior of species in nature are called *swarm intelligence algorithms*. Typical examples of these algorithms are ant colony optimization [20, 23, 24], particle swarm optimization [28, 64, 93], and artificial bee colony algorithms [61–63]. Swarm intelligence algorithms are not classified as evolutionary algorithms, but they have a similar framework for optimizing and solving problems. Hence, swarm intelligence and evolutionary algorithms are often discussed together in conferences[*2] and journals[*3].

The outline of a typical evolutionary algorithm is shown in Algorithm 2.1. Evolutionary algorithms use a population of candidate solutions to solve optimization problems. In general, a candidate solution is called an *individual* or a *phenotype*. When an individual is encoded in the space where the evolutionary search takes place, it is called a *genotype*. Each individual is evaluated using the *fitness function* or also called the *objective function*. This function assigns a fitness value that indicates the quality of the individual. In other words, the fitness value indicates how well the individual is close to solving the optimization problem. For each generation, individuals compete to reproduce offspring. Individuals with higher fitness values have more chance of becoming parents. Offspring are generated by combining parts of the parent genotypes. This genetic operator is called *recombination* or *crossover*. Another

[*2] E.g., the IEEE Congress on Evolutionary Computation (CEC), the Genetic and Evolutionary Computation Conference (GECCO), etc.

[*3] E.g., IEEE Transactions on Evolutionary Computation, Evolutionary Computation (MIT Press), Swarm and Evolutionary Computation (Elsevier), etc.



(a) Feedforward neural network. **(b)** Recurrent neural network.

Fig. 2.1. Basic structures of artificial neural networks.

genetic operator is called *mutation*, which applies a slight perturbation to a genotype of offspring. The offspring individuals are evaluated, and then those with high fitness values are selected to survive to the next generation. These processes are repeated until the terminal condition is satisfied, e.g., finding the individual with a well enough fitness value or reaching the maximum number of generations.

2.1.2 Neuroevolution

Neuroevolution is a machine learning technique that applies evolutionary algorithms to optimize artificial neural networks [33, 74, 105]. Historically, not all work combining neural networks and evolutionary computation was called neuroevolution. Hence, neuroevolution is sometimes simply called *evolving artificial neural networks* [122]. However, the term neuroevolution is gradually being used in the field of machine learning.

Artificial neural networks are computational models that are inspired by a biological brain. An artificial neural network consists of a collection of nodes called neurons. Similar to a biological brain, neurons could transmit a signal to other neurons by given synaptic connections. Each connection has a synaptic weight value that could increase or decrease the strength of the signal.

The most commonly used neural networks have layered structures with the input, hidden, and output layers. The examples of basic structures of neural networks are shown in Fig. 2.1. The most standard architecture is called a *feedforward neural network*, in which signal information only flows in one direction, from the input to the output layer (see also Fig. 2.1(a)). Feedforward neural networks are often used in pattern classification; however, they are not good at detecting or producing temporal sequences. A simple way for providing

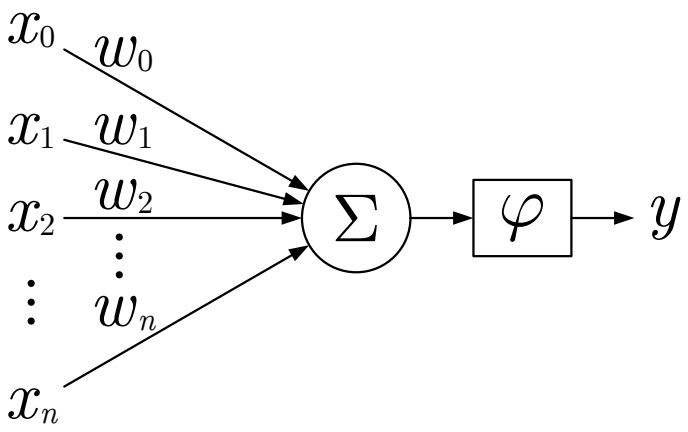


Fig. 2.2. Artificial neuron model.

temporal dynamics to neural networks is to construct feedback connections. These neural networks with feedback connections are called *recurrent neural networks* (see also Fig. 2.1(b)). In addition, neural networks with multiple hidden layers are called *deep neural networks*, which have attracted considerable interest in recent years.

An example of an artificial neuron is illustrated in Fig. 2.2. When n inputs are fed into the neuron, the output y could be calculated as follows:

$$y = \varphi\left(\sum_{i=0}^{n} w_i x_i\right), \tag{2.1}$$

where x_i is the ith input signal, w_i is the synaptic weight value of the input x_i, and $\varphi(\cdot)$ is the activation function. The most commonly used activation functions are sigmoid and hyperbolic tangent functions. In the field of deep learning [45, 73], a rectified linear function is also used as the activation function. Typically, each neuron has a *bias* that shifts the activation function by adding a constant value. The bias could be implemented to the neuron by simply setting one input signal having a constant value.

The weight values of the neural network are adjusted in the training process. In general, neural networks are trained in supervised learning using a gradient-based learning algorithm. In contrast, neuroevolution optimizes the weight values through an evolutionary algorithm. Neuroevolution allows learning without explicit targets or gradient information. Typically in neuroevolution, the topological structure of the neural network is fixed, and the vector of the synaptic weight values is encoded to the genotype. On the other hand, alternative methods are proposed to evolve both the values of the synaptic weights and the topological structure of the neural network [106]. These methods are called Topology and Weight Evolving Artificial Neural Networks (TWEANNs).

2.1.3 Evolutionary Robotics Approach for Designing Controllers

The majority of the studies on evolutionary robotics follow the classic evolutionary algorithm scheme. When designing controllers using the evolutionary robotics approach, the parameters of the robot controller are encoded into the genotype. A typical evolutionary robotics approach uses neuroevolution, in which the vector of the synaptic weight values of the neural network controller is encoded into the genotype. In the process of evaluation, each genotype is decoded into the controller and implemented to the robot. The fitness value is determined based on the performance of the robot or the achievement of the task.

The emergent behavior of a robot ultimately depends on the formulation of fitness functions. In other words, the settings of fitness functions determine whether or not the controller successfully evolved to make the robot perform the desired behavior. Nelson et al. [88] have classified the fitness functions by considering the prior knowledge required for designing them. The prior knowledge could reflect the level of truly novel learning that has been accomplished. In other words, the robot has to autonomously and adaptively generate behavior through artificial evolution if there is less prior knowledge in fitness functions. On the other hand, the more prior knowledge required, the more likely the robot behavior is guided by the designer. The fitness functions in evolutionary robotics could be classified as follows [88].[*4]

Behavioral Fitness Functions

Behavioral fitness functions are defined by what a robot is doing and how it is doing. These are task-specific hand-formulated functions and generally include several subfunctions that measure simple behaviors or low-level sensor-actuator mappings. These functions measure how a robot is behaving and not what it has accomplished. For example, to make a robot move around the environment while avoiding obstacles, fitness functions should be designed to be maximized if the robot turns when its front sensors are stimulated. The fitness functions are not directly designed to avoid obstacles because the robot evolved to turn if it detects an obstacle in front. However, as a result, the robot could generate behavior to avoid obstacles. Behavioral fitness functions require a high-level prior knowledge of the tasks addressed by the robot and the desired behavior. In addition, the designer has to determine functions that are expected to produce the desired robot behaviors.

[*4] Nelson et al. [88] have classified fitness functions in evolutionary robotics into seven classes. *Training data fitness functions* are designed to minimize the error using a training data set, similar to a gradient-based learning algorithm. Training data fitness functions are less likely to be used in designing controllers for robotic swarms. *Functional incremental fitness functions* and *environmental incremental fitness functions* are used when evolving controllers with *incremental evolution*, and *competitive and co-competitive fitness selection* is used in *co-evolution*. These fitness functions are used in special cases of evolutionary robotics frameworks. Therefore, these fitness functions are excluded in this thesis.

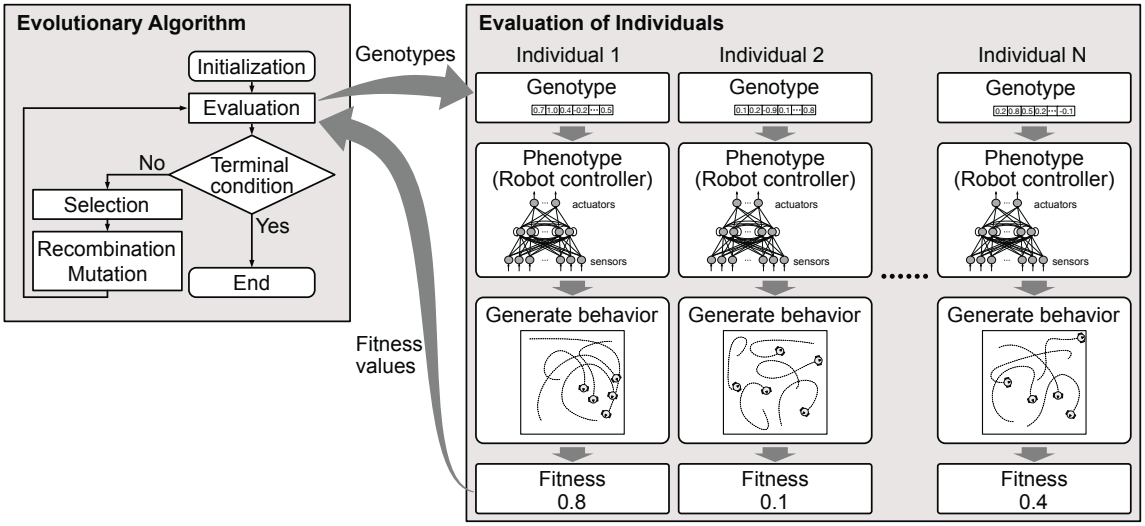


Fig. 2.3. Outline of an evolutionary robotics approach.

Aggregate Fitness Functions

Aggregate fitness functions are defined by the success or failure of the task addressed by a robot. These functions are set without considering how the task was accomplished. These functions could reduce the injection of the designer's bias on the behavior emerged by the robot. Also, aggregate fitness functions require very low prior knowledge. However, when the task addressed by a robot is too complex, all individuals within the evolutionary algorithm perform equally poorly, causing evolution to drift in an uninteresting region of the search space. This problem is referred to as the *bootstrap problem* [101].

Tailored Fitness Functions

Tailored fitness functions are the combination of behavioral and aggregate fitness functions. This type of fitness function is most often used in the field of evolutionary robotics. Tailored fitness functions require a moderate level of prior knowledge.

2.2 Evolutionary Swarm Robotics

The evolutionary robotics approach is a promising method for designing a controller for a robotic swarm. In general, a swarm robotics system is composed of homogeneous robots. Therefore, an identical controller is copied to all of the robots within the robotic swarm. Similar to a typical evolutionary robotics technique, the parameters of the controller are encoded into the genotype. In the evaluation of individuals, the genotype is decoded into the controller and copied to each robot. The outline of a typical evolutionary robotics approach is illustrated in Fig. 2.3.

Typical evolutionary robotics approaches use neural networks as robot controllers. The

topological structure of the neural network is fixed, and the values of the synaptic weights are optimized via an evolutionary algorithm. Several collective behaviors have been developed with this approach, e.g., aggregation [103, 114], flocking [2], path formation [54, 104], and cooperative transport [1, 46, 120]. As an alternative method, there is an approach for evolving both the synaptic weight values and the topological structure of the neural network [106]. Few studies have been reported using this approach for designing a controller for a robotic swarm (e.g., [26, 44]). In recent years, neural networks with multiple hidden layers, which are also called *deep neural networks*, have attracted considerable interest [45, 73]. The combination of deep neural networks and neuroevolution is called *deep neuroevolution* [109]. The experiment of developing controllers for a robotic swarm using deep neuroevolution is reported in [87]. Other than the approaches using neural networks, some methods use finite state machines as the controllers, and those parameters and topologies are evolved [40, 66].

In evolutionary robotics, individuals of the evolutionary algorithm are evaluated by decoding them into controllers and then generating robot behavior. Typically, evolutionary algorithms use dozens to hundreds of individuals, and consequently, the evaluation process requires a considerable amount of time. Therefore, evolutionary robotics often utilizes computer simulations to accelerate and parallelize the evaluation process. The technique that uses computer simulations is referred to as *offline evolution* [8]. However, when controllers developed in computer simulations are transferred to the physical robots, the robots do not behave in the same way as simulations. This problem is called the *reality gap*, which is one of the open issues in evolutionary robotics [59, 101]. As for now, there is no general method to overcome the reality gap problem; however, there are some promising approaches to reduce the gap (e.g., [67, 78]). One way to address this problem is *online evolution*, in which an evolutionary algorithm is implemented inside the robots and evolves controllers while the robots perform their tasks [11, 39]. Online evolution is still in its infancy, and only a few studies have been reported using this approach (e.g., [11, 102, 119]).

2.3 Conclusions

This chapter presented a brief introduction to automatic design methods in swarm robotics. The automatic design method develops a controller for a robotic swarm by casting the design problem into an optimization problem. The evolutionary robotics approach is a widely used automatic design method that utilizes an evolutionary algorithm to optimize the parameters of the robot controller. The typical evolutionary robotics approach utilizes neuroevolution to develop the robot controller.

This thesis focuses on offline evolution to design controllers for robotic swarms. In this thesis, aggregate fitness functions are employed to reduce restrictions in behaviors that may emerge in robotic swarms. Chapters 3 and 4 show how the typical evolutionary robotics

approach could be applied to generate collective behavior that is difficult to be developed manually by the designer. The evolutionary robotics approach using echo state networks, which are an alternative to the traditional recurrent neural networks, is proposed in Chapter 6. Chapter 7 focuses on TWEANN algorithms for designing controllers for robotic swarms.

Chapter 3

Emergence of Collective Cognition in a Cooperative Foraging Task

This chapter focuses on the collective cognition performed by a robotic swarm. In biological swarms, individual cognition is enhanced by communication and cooperation with other individuals. A single individual only has a limited cognition ability; however, it could be extended beyond its capabilities as a swarm. Benefitting from collective cognition, biological swarms can exhibit sophisticated collective behavior and decision-making processes [17, 115]. On the other hand, the robotic swarms have to rely on collective cognition more than biological swarms when considering the limitation in sensory capabilities and the cost of each robot. Moreover, the collective cognition of a robotic swarm has potential applications to tasks such as locating targets in an unknown environment (e.g., undersea or planetary explorations) or search-and-rescue operations in a disaster area. Despite the importance of collective cognition, only a few studies have addressed this problem in the field of swarm robotics [29, 98, 117].

The research on collective cognition in robotic swarms mainly applies the behavior-based design method to design the controllers. More specifically, the robots are controlled by a finite state machine that is designed manually. For example, collective cognition is studied in the aggregation task with two different sized target areas [98]. In this task, robots have to form aggregates with the proportion that corresponds to the target area size. Robots cannot measure the size of the area. However, they can achieve the task by communicating their local perception of the environment with nearby robots. In another related work, a robotic swarm achieves a decision-making task by collective cognition [117]. In this task, the environment is covered with two different features and determines which feature covers the most. Robots

can only perceive the feature of the environment by their ground sensor. However, they can achieve the task by local communication among robots to perform collective cognition.

This chapter demonstrates the evolutionary robotics approach for designing controllers to perform collective cognition in a cooperative foraging task. In this task, the robots have to distinguish between two types of objects, i.e., food and poison objects, scattered in the field by performing collective cognition. At the same time, robots have to transport food objects while avoiding poison objects. It is worth noting that a single robot cannot distinguish between food and poison objects. Therefore, the robotic swarm has to perform collective cognition to distinguish between two types of objects. The experiments are constructed in computer simulations. Additional experiments are conducted to examine the scalability and flexibility of the developed controllers.

The remainder of this chapter is organized as follows. Section 3.1 describes the settings of the foraging task and the evolutionary robotics approach. Section 3.2 discusses the results of the experiments. Finally, Section 3.3 concludes this chapter.

3.1 Settings of the Experiments

The objective of the task is that the robots should distinguish between the two kinds of objects, namely food and poison objects, and transport only food objects to the nest. A single robot cannot distinguish between food and poison objects by itself. Therefore, the robotic swarm has to perform collective cognition to distinguish between the two objects. Additionally, food and poison objects are too heavy for a single robot to move. Hence, robots have to cooperate with each other to accomplish the task. The experiments are carried out in computer simulations using the *Box2D* physics engine [15]. The remainder of this section describes the settings of the environment, the robot, and the evolutionary robotics approach used in the experiments.

3.1.1 Collective Foraging Task with Poison Objects

The simulation environment has a regular octagon field, as shown in Fig. 3.1. A circle-shaped nest with a radius of 15 m is located in the center of the field. At the beginning of the simulations, robots are positioned inside the nest. The initial positions of the food and poison objects are randomly determined. There are always five food objects and five poison objects in the field. A new food or poison object will be generated with a random position when it is transported to the nest. The radius of food and poison objects are set to 5.0 m and 2.5 m, respectively. The food and poison objects are set to have the same weight; in particular, at least four robots are required to move an object. Since a robot does not have the ability to distinguish between the two objects, the only difference between them is in the size (the sensor settings of the robot are described in Section 3.1.2).

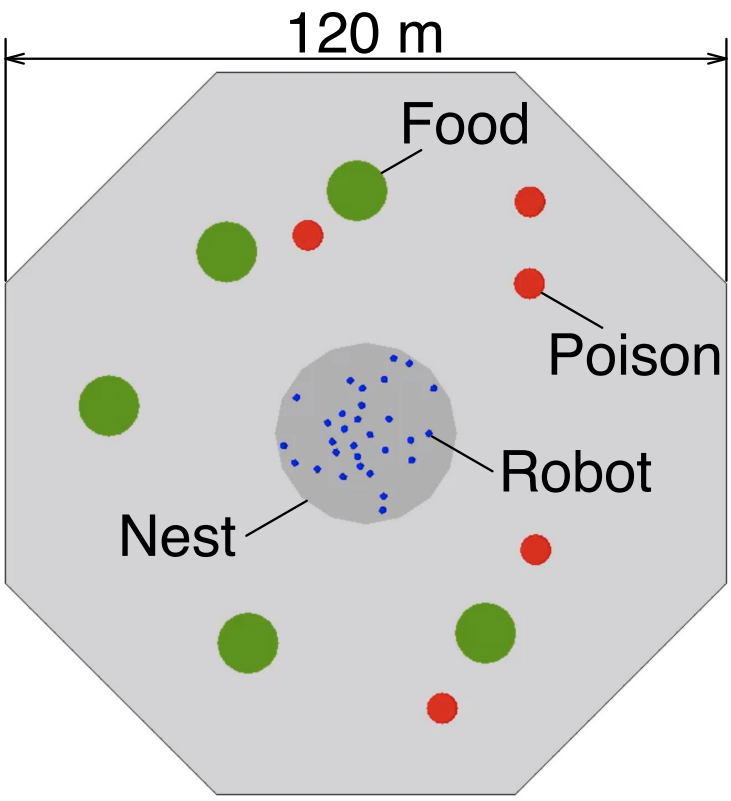


Fig. 3.1. Collective foraging task with poison objects.

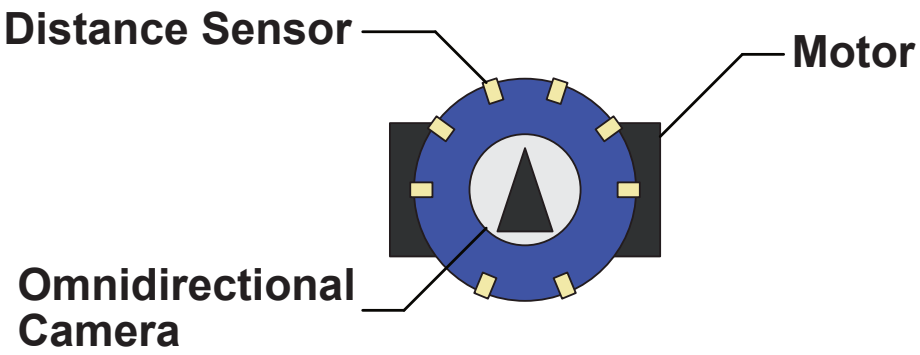


Fig. 3.2. Specifications of the robot.

3.1.2 Settings of the Robot

The specifications of the robot are shown in Fig. 3.2. Each robot is composed of eight distance sensors, an omnidirectional camera, an artificial neural network controller, and two motors to rotate the left and right wheels. The range of the distance sensor is set to 3.0 m, detecting the nearest objects, robots, or walls. Each distance sensor returns a value that corresponds to the distance to the detected item. The range of the omnidirectional camera is set to 15 m, gathering one input for the distance and two inputs for the relative angle,

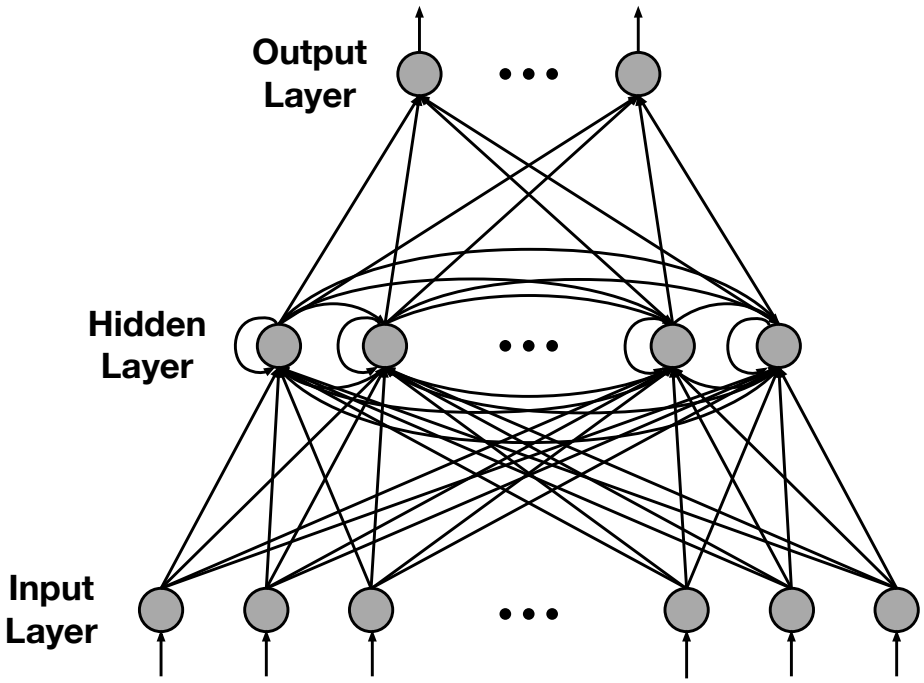


Fig. 3.3. Structure of the robot controller. The controller is represented by the fully-connected recurrent neural network. The nodes in the hidden layer are connected with each other and also have self-connections.

represented by the sine and cosine of the angle, of the followings:

- The nearest food or poison object.
- The second nearest food or poison object.
- The nearest robot.
- The second nearest robot.
- Angle of the nest (without distance).

Three inputs are collected for each item; except for the nest, only two inputs are collected. It is important to note that a single robot cannot distinguish between food and poison objects, because the omnidirectional camera treats food and poison objects as the same *objects*. Each robot is governed by a three-layered recurrent neural network as shown in Fig. 3.3. The first layer gathers sensory inputs from the distance sensors and the omnidirectional camera. In total, twenty-two inputs are fed into the input layer, i.e., eight inputs from the distance sensors and fourteen inputs from the omnidirectional camera. The hidden layer is composed of twenty nodes with recurrent connections including self-connections. Outputs of the third layer control the two motors directly. The logistic function is employed as the activation function for the hidden and the output layer.

3.1.3 Settings of Evolutionary Robotics Approach

The covariance matrix adaptation evolution strategy is employed to optimize the synaptic weights of the controller. The covariance matrix adaptation evolution strategy is widely known as an efficient population-based black-box optimization algorithm. In this algorithm, each candidate solution is represented by a real-valued vector. While the classic evolution strategies sample the offspring for the next generation from a normal distribution, the covariance matrix adaptation evolution strategy samples offspring the following equation:

$$\boldsymbol{x}_i \sim \boldsymbol{m} + \sigma \mathcal{N}_i(0, \boldsymbol{C}), \qquad \text{for } i = 1, 2, ..., \lambda, \tag{3.1}$$

where $\boldsymbol{m} \in \mathbb{R}^n$ is the mean vector (search point) in the last generation and $\sigma \in \mathbb{R}_{>0}$ is the so-called step size. $\mathcal{N}_i(0, \boldsymbol{C})$ is the multivariate normal distribution with zero mean and $\boldsymbol{C} \in \mathbb{R}^{n \times n}$ is the covariance matrix which determines the shape of the distribution ellipsoid. Here, i is the index of candidate solutions, n represents the dimension of the candidate solution, and λ is the population size.

Compared to classical evolution strategies algorithms, the covariance matrix adaptation evolution strategy performs a more efficient search by using covariance matrix adaptation and cumulation on the evolution path. Additionally, the covariance matrix adaptation evolution strategy provides recommendation values for the hyperparameters. Therefore, only the population size λ and the initial step size σ are set depending on the objective function. More details about the covariance matrix adaptation evolution strategy can be found in [49–51].

At the first generation of artificial evolution, a population of λ candidates is initialized with random values. Each candidate controller is copied to N robots and evaluated 10 times in the collective foraging task. The average value over the 10 trials is used as the fitness value of the candidate controller. The fitness function F is composed of two parts, a reward for transporting food objects and a penalty for poison objects, which is defined as follows:

$$F = \sum_i d_{\text{food},i} - \sum_j d_{\text{poison},j}\,, \tag{3.2}$$

where $d_{\text{food},i}$ is the distance shortened between the ith food object and the center of the nest, and $d_{\text{poison},j}$ is the distance shortened between the jth poison object and the nest. Each trial of the collective foraging task lasts for 9000 time steps (0.02 s for each time step, in total 180 s), and subsequently, the fitness value is calculated by Eq. (3.2). The candidates with higher fitness values are selected to produce the next population. These processes are repeated until the maximum generation. In this chapter, the population size of the covariance matrix adaptation evolution strategy algorithm is set to $\lambda = 300$, the initial step size is set to $\sigma = 0.2$, and the maximum generation is set to 2000.

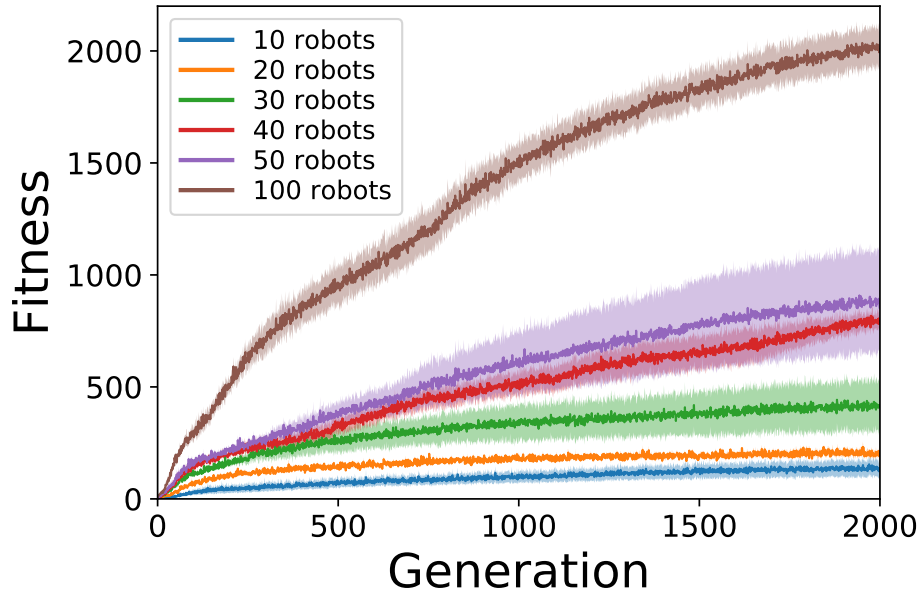


Fig. 3.4. Transitions of the best fitness value. The solid line indicates the mean over the five trials, and the shaded area shows the standard error of the mean. The color of the plots shows the experiment settings with the different number of robots.

3.2 Results and Discussion

The experiments are performed with $N = 10$, 20, 30, 40, 50, and 100 robots. Five evolutionary trials are executed for each experiment. After running the evolutionary process, further experiments are conducted to test the scalability of the developed controllers. In scalability experiments, the performance of the controllers is tested in different numbers of robots as they are developed. For instance, the controller developed in the environment with 10 robots is tested in environments with 20, 30, 40, 50, and 100 robots. In addition, the experiments are conducted by changing the radius of the poison objects to examine how well the robots can distinguish between the foods and the poison objects. In these experiments, the radius of the poison objects is changed to 3.5 m and 4.5 m. Also, the food and poison objects are adjusted to have the same weight. The flexibility experiments are performed by testing the developed controllers in the environment with different poison sizes.

The transitions of the fitness values of the evolutionary trials are shown in Fig. 3.4. The fitness value plateaus in experiments with 10, 20, and 30 robots, whereas the fitness keeps increasing in experiments with 40, 50, and 100 robots even with 2000 generations, as shown in Fig. 3.4. The performance of the best controller developed in each experiment is re-evaluated 100 times. The results of the re-evaluation are shown in Fig. 3.5. As can be seen from Fig. 3.5, the performance increases as the number of robots increases. The examples of the behavior obtained with 20, 30, and 100 robots are shown in Figs. 3.6, 3.7, and 3.8, correspondingly. In

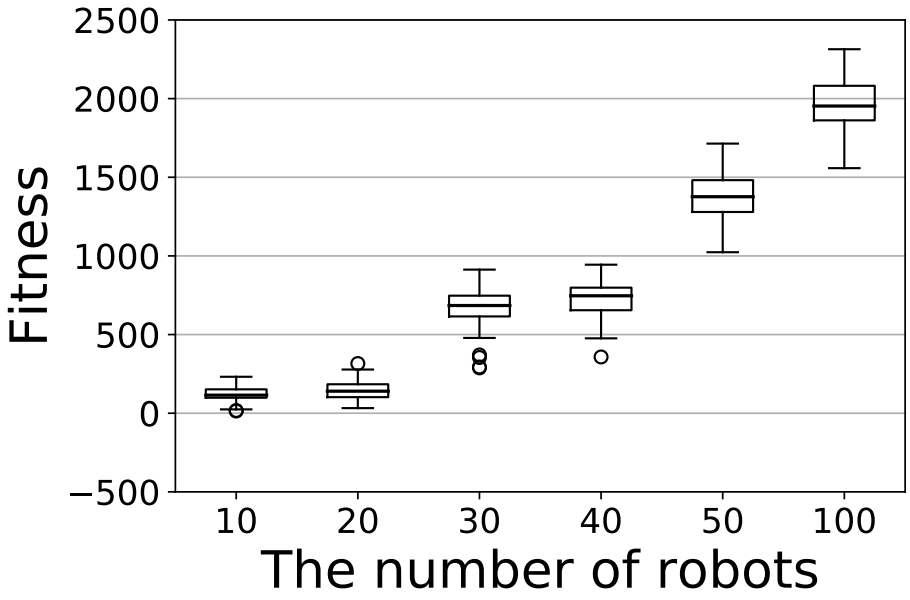
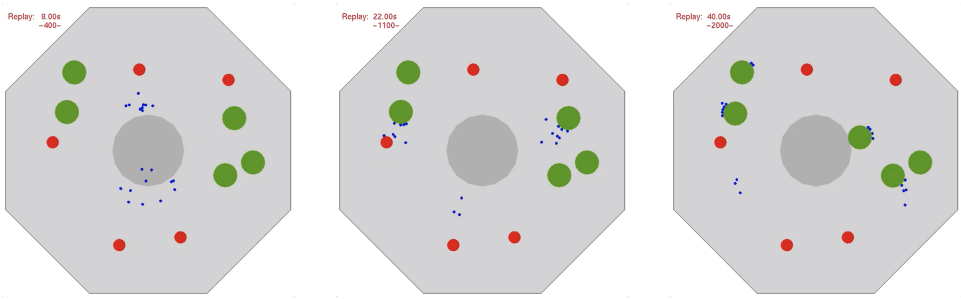
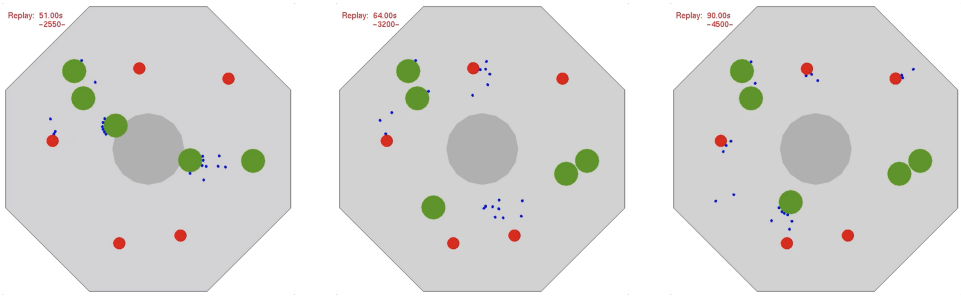


Fig. 3.5. Results of the re-evaluation of the best controller developed in experiments over 100 trials.

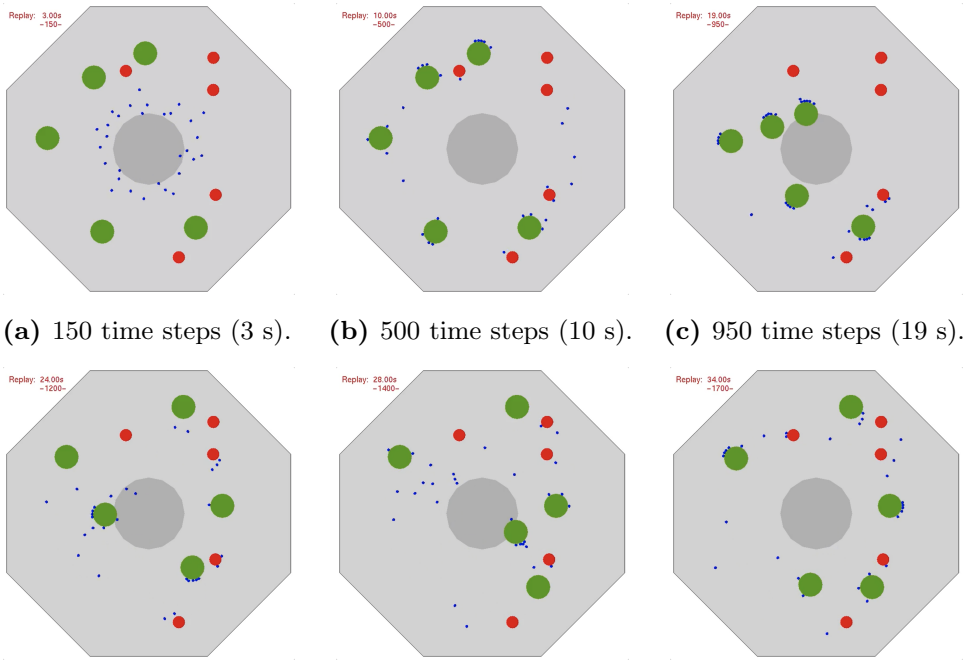


(a) 400 time steps (8 s). **(b)** 1100 time steps (22 s). **(c)** 2000 time steps (40 s).



(d) 2550 time steps (51 s). **(e)** 3200 time steps (64 s). **(f)** 4500 time steps (90 s).

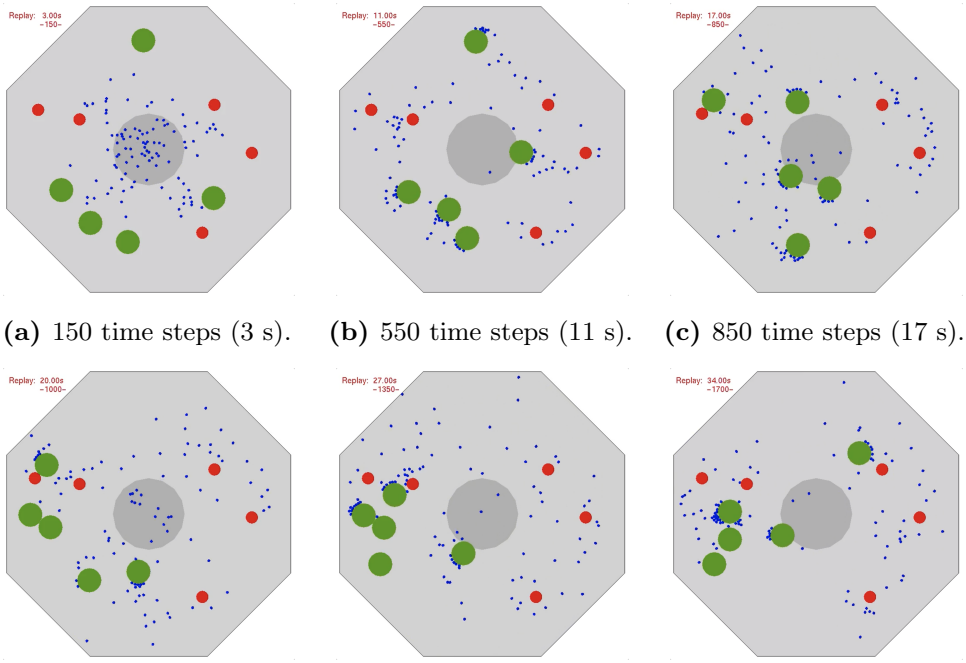
Fig. 3.6. Snapshots of the behavior observed using the controller developed in the experiment with 20 robots.



(a) 150 time steps (3 s). **(b)** 500 time steps (10 s). **(c)** 950 time steps (19 s).

(d) 1200 time steps (24 s). **(e)** 1400 time steps (28 s). **(f)** 1700 time steps (34 s).

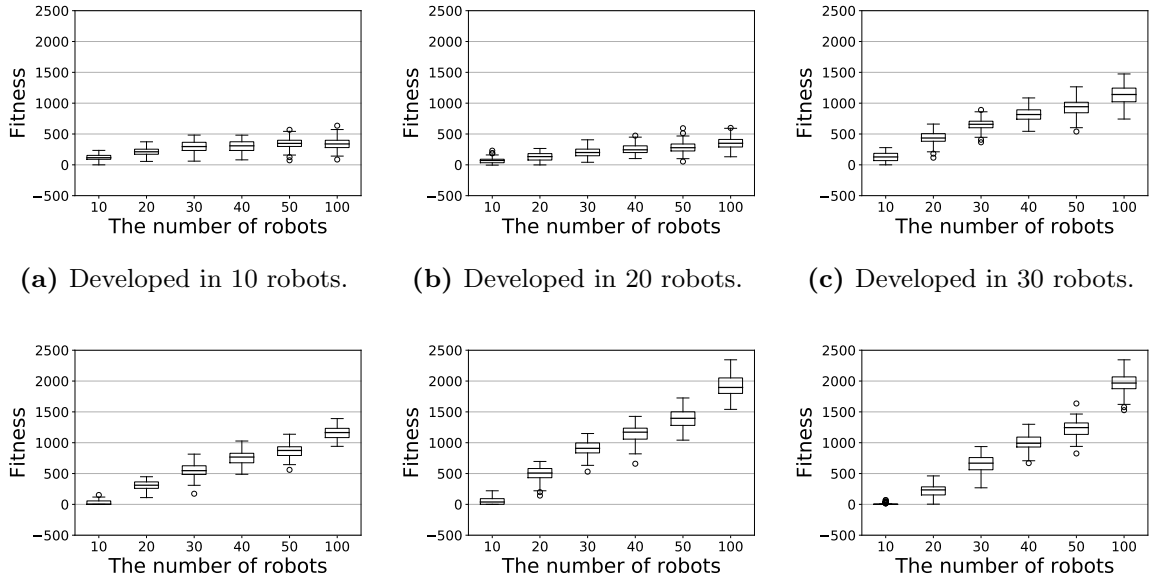
Fig. 3.7. Snapshots of the behavior observed using the controller developed in the experiment with 30 robots.



(a) 150 time steps (3 s). **(b)** 550 time steps (11 s). **(c)** 850 time steps (17 s).

(d) 1000 time steps (20 s). **(e)** 1350 time steps (27 s). **(f)** 1700 time steps (34 s).

Fig. 3.8. Snapshots of the behavior observed using the controller developed in the experiment with 100 robots.



(a) Developed in 10 robots. **(b)** Developed in 20 robots. **(c)** Developed in 30 robots.

(d) Developed in 40 robots. **(e)** Developed in 50 robots. **(f)** Developed in 100 robots.

Fig. 3.9. Results of the scalability experiments of the controller developed in the experiments with (a) 10, (b) 20, (c) 30, (d) 40, (e) 50, and (f) 100 robots.

experiments with 10 or 20 robots, the artificial evolution developed strategies in which the robots are more likely to move as a group to transport only food objects (see also Fig. 3.6). On the other hand, for experiments with 30, 40, 50, or 100 robots, the developed strategies allow the robots to act more independently for an efficient exploration (see Figs. 3.7 and 3.8).

The results of the scalability experiments are as shown in Fig. 3.9. As can be seen from Figs. 3.9(a) and 3.9(b), the strategies developed in experiments with 10 or 20 robots, which perform the task by forming groups, show poor scalability to a larger number of robots. There is no remarkable increase in fitness values by increasing the number of robots. These results are due to the lack of parallelization to execute the task. The strategies developed in experiments with more robots, which perform the task more independently, show better scalability than the strategies that are generated with a smaller number of robots. However, the strategies to act more independently do not seem to show scalability in situations with 10 robots, especially for the controllers developed in 40, 50, and 100 robots (see also Figs. 3.9(d), 3.9(e), and 3.9(f)). The strategy to act independently makes robots spread out within the environment, which leads to efficient exploration and parallelization of the task. In situations with a smaller number of robots, however, makes robots perform cognition more independently. Besides, a smaller number of robots spreading out within the environment makes it difficult to achieve the task because at least four robots have to get together to move an object. On the other hand, the controller developed in 50 robots seems to show high scalability by obtaining high fitness values also in environments with 20, 30, and 40 robots.

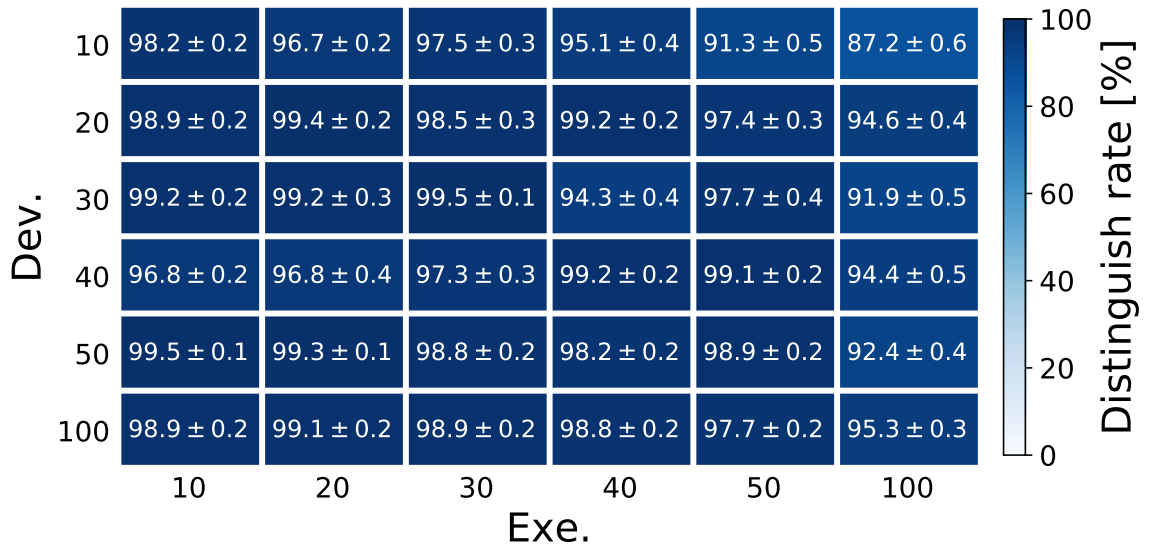


Fig. 3.10. Distinguish rates (%) in the scalability experiments with the standard error of the mean over 100 trials. The rows of the figure represent the number of robots in the development environments (Dev.) and the columns represent the execution environments (Exe.).

For further discussion, the distinguish rate R is calculated by the following equation:

$$R = \frac{\sum_i d_{\text{food},i}}{\sum_i d_{\text{food},i} + \sum_j d_{\text{poison},j}}, \tag{3.3}$$

where $d_{\text{food},i}$ is the distance shortened between the ith food object and the center of the nest, and $d_{\text{poison},j}$ is the distance shortened between the jth poison object and the nest. This equation indicates that the distinguish rate R equates to the total transport distance of the food objects divided by the transport distance for all objects. The distinguish rates for the scalability experiments are as shown in Fig. 3.10. As can be observed from Fig. 3.10, the distinguish rates were kept at high values even in the experiments that showed low scalability.

The experiments with different poison sizes are performed to examine how well the robots can distinguish between the food and the poison objects. The radius of the poison object is increased to 3.5 and 4.5 m to make it difficult to distinguish between the two object types. The experiments are performed with 30 robots. Five evolutionary trials are executed for both experiments with the poison size of 3.5 and 4.5 m. The performance of the best controller developed in the experiments with the different poison sizes is re-evaluated 100 times. The results of the re-evaluation are shown in Fig. 3.11. For comparison, the re-evaluation using the 2.5 m poison size is also shown in Fig. 3.11. As can be seen from Fig. 3.11, the performance decreases as the poison size increases. In cases of the poison size with a radius of 4.5 m, the

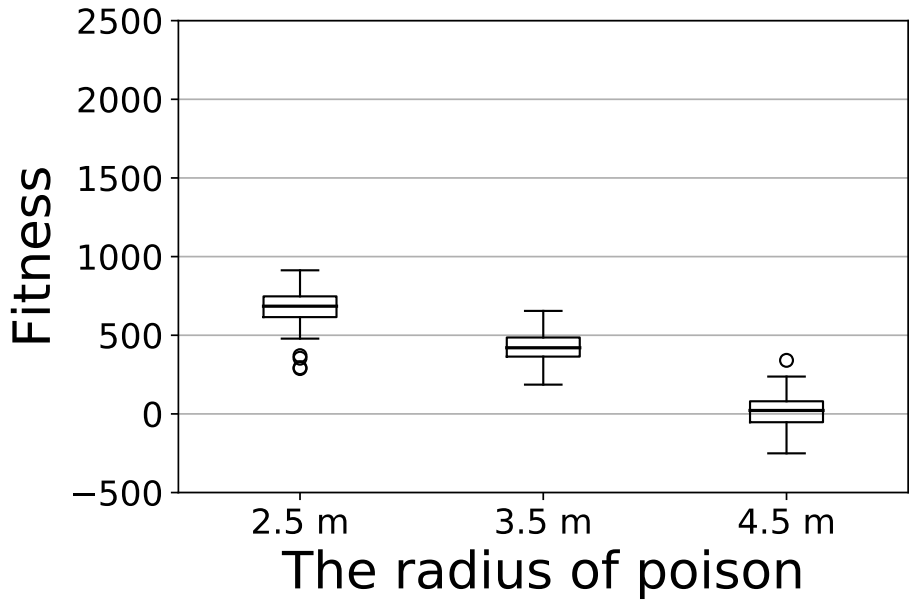
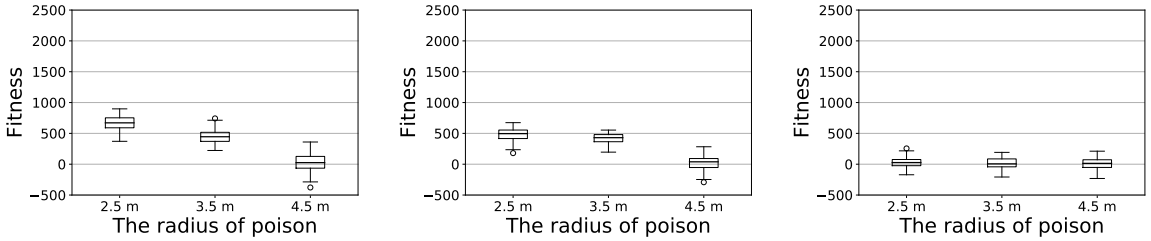


Fig. 3.11. Results of the re-evaluation with the best controller developed in experiments with the different poison sizes.



(a) Developed in 2.5 m radius poison objects.

(b) Developed in 3.5 m radius poison objects.

(c) Developed in 4.5 m radius poison objects.

Fig. 3.12. Results of the flexibility experiments of the controller developed in the experiments with the poison objects with the radius of (a) 2.5 m, (b) 3.5 m, and (c) 4.5 m.

robots could not distinguish between the two objects.

The results of the flexibility experiments are shown in Fig. 3.12 and the distinguish rates are shown in Fig. 3.13. As can be seen from the results, the controller developed in easier environments (with larger differences in size between the two objects) exhibit more flexibility, e.g., the controller developed with the poison size of 2.5 m radius also performed well in the environments with 3.5 m poison radius. Whereas in situations with the poison size of 4.5 m radius, the robotic swarm could not distinguish between the two objects regardless of the environment in which the controllers were developed (see also Fig. 3.12 and 3.13). These results are due to the bootstrap problem [101] that occurs when all of the controllers

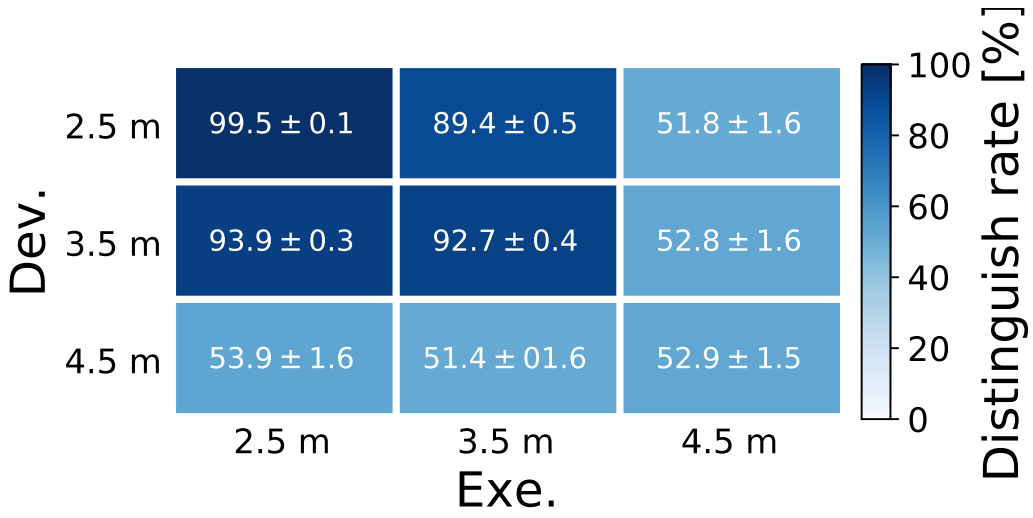


Fig. 3.13. Distinguish rates (%) in the flexibility experiments with the standard error of the mean over 100 trials.

in the earlier stages of evolution perform equally poorly, which drifts evolution towards a local optimum.

3.3 Conclusions

This chapter showed how the evolutionary robotics approach could be applied to generate collective cognition by robotic swarms in the foraging task. In this task, collective cognition was performed by the robotic swarm to distinguish between two types of objects and cooperatively transport one type of them. The covariance matrix adaptation evolution strategy algorithm was adopted to develop controllers for the robotic swarm under different conditions. Additional experiments were performed to examine the scalability and flexibility of the developed controllers. The results showed that collective cognition was successfully developed, which allows the robots to transport only food objects. However, it was difficult for robotic swarms to distinguish between the two objects if there was much less difference between them.

Chapter 4

Emergence of Behavioral Specialization in a Path-formation Task

Task allocation, also known as division of labor, is one of the collective behaviors that have attracted considerable interest in both biological and engineering viewpoints. Task allocation is a concept to make individuals within a swarm specialized to different tasks or behavior. Task allocation makes a robotic swarm perform tasks more efficiently than a single high-performance robot. In addition, task allocation could enhance fault tolerance, scalability, and flexibility of a robotic swarm.

Most research on task allocation in swarm robotics address a foraging task, which could intuitively decompose the main task into subtasks (e.g. [12, 70, 71, 79]). Moreover, a robot controller to perform task allocation is mainly designed using a behavior-based approach. In these studies, finite state machines with response thresholds to change states [70, 79] or probabilistic transitions between states [12, 71] are used as the robot controllers. However, there are limitations to finite state machines; i.e., they can only consider a restricted number of behavior states, and each state must be predefined [32].

This chapter focuses on a special case of task allocation, where the main task is difficult to decompose into predefined subtasks. In particular, this chapter aims to develop a robot controller to manage congestion by task allocation. Generally, robotic swarms are conducted to have high redundancy in the number of robots. However, when multiple robots gather in a spatially limited environment, robots tend to interfere with each other, which decreases the performance of a robotic swarm [75]. This congestion is a critical issue in the navigation of robotic swarms because a large number of robots are required to move toward the same target simultaneously [82]. Despite the importance of managing congestion in a robotic swarm, only a few studies have addressed this problem.

This chapter demonstrates how the evolutionary robotics approach could be applied to

design a robot controller that exhibits task allocation to manage congestion. The robot controller is developed and tested in a path-formation task. In this task, a robotic swarm aims to develop a collective path of robots and navigate between two landmarks. The experiments are conducted by varying the number of robots. This chapter shows that the robots specialize their behavior in a situation with larger swarm sizes to mitigate congestion. In addition, the experiments are conducted to investigate how the robotic swarm develops a strategy to manage congestion within the evolutionary process.

The rest of this chapter is organized as follows. Section 4.1 describes the task addressed in this chapter and the robot settings used in the experiments. Section 4.2 presents the settings of the evolutionary robotics approach. The results of the experiments by varying the number of robots are shown in Section 4.3. Section 4.4 further discusses how the evolutionary robotics approach developed controllers to exhibit strategy to mitigate congestion. Finally, Section 4.5 concludes this chapter.

4.1 Settings of the Path-formation Task

This section describes the task to be addressed by a robotic swarm, along with the settings of the robot. A path-formation task is one of the fundamental tasks addressed in the study of swarm robotics [4, 10, 104]. In this task, the goal of a robotic swarm is to develop a collective path of robots and navigate between two landmarks. The experiments are carried out in computer simulations using the *Box2D* physics engine [15].

4.1.1 Task Environment

The environment for the path-formation task is shown in Fig 4.1. The environment consists of a square-shaped arena that is surrounded by walls with two landmarks placed inside. Each landmark has a colored LED light source and a target area with a radius of 0.5 m. A robot is considered to have arrived at the landmark when the robot travels inside the corresponding target area. The robotic swarm should develop a path between the two target areas and visit them alternately.

4.1.2 Robot Settings

The settings of the robot are illustrated in Fig 4.2. The robot has a circular body with a radius of 0.1 m and moves with a two-wheeled differential drive method with a maximum velocity of approximately 0.2 m/s. The robot has colored LEDs, an omnidirectional camera, seven distance sensors, and a ground sensor. The LEDs around the robot emit blue and red lights from the front and rear sides of the robot, respectively. The activation of the front and rear LEDs are controlled independently depending on the outputs from the controller. The LED light source of both landmarks always emits the red color, which is the same color

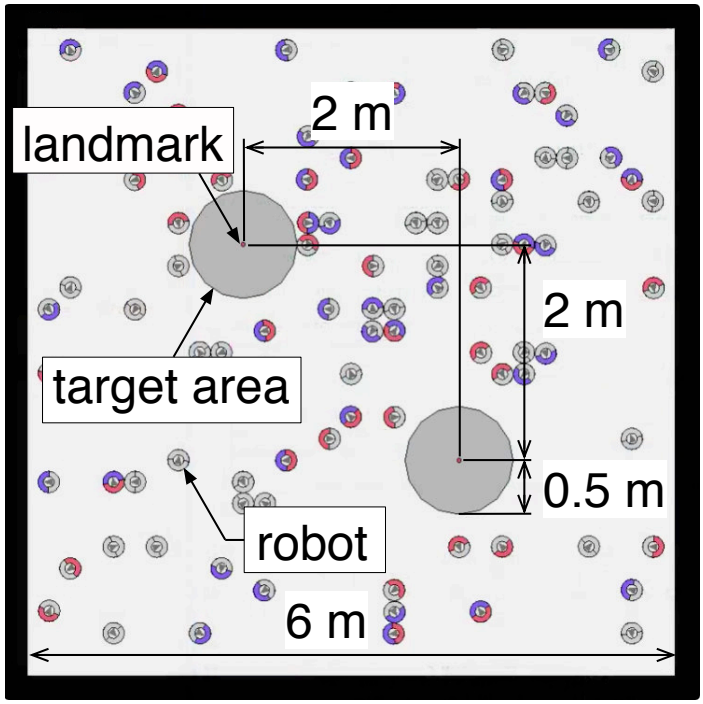


Fig. 4.1. Snapshot of the environment of the path-formation task. The two gray circles indicate the target areas, with a landmark placed in the center of them. The color of the robots shows the activation of the LEDs, with the light gray color indicating the deactivation of the corresponding LEDs.

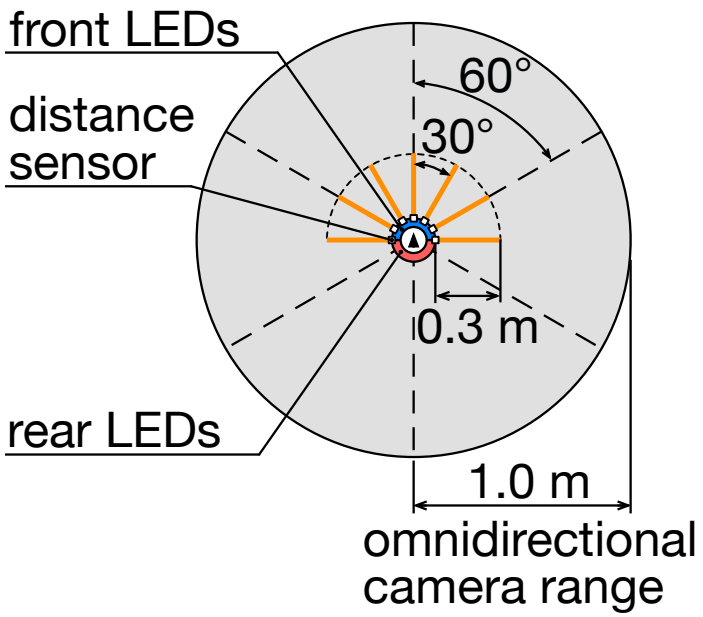


Fig. 4.2. Settings of the robot. Distance sensors are attached to the front side of the robot with an interval of $\pi/6$ radians. The vision of the omnidirectional camera is divided into six circular sectors with a central angle of $\pi/3$ radians.

as the robot's rear LED lights.

The visual input from the omnidirectional camera is converted to detect the existence of light sources from the colored LEDs. The circular-shaped sensor range of the omnidirectional camera is divided into six equal slices, as can be seen in Fig 4.2. The omnidirectional camera only detects the existence of colored LED lights for each slice. The process for producing sensor values from the omnidirectional camera is done independently for each colored light source (i.e., blue and red colors). Each slice in the omnidirectional camera returns a value of 1 if the corresponding color lights have been detected and 0 otherwise. In total, twelve binary inputs are obtained from the omnidirectional camera.

Distance sensors are attached to the front side of the robot with an angular interval of $\pi/6$ radians. The distance sensor can detect walls and other robots within the sensor range. The sensor value from the distance sensor is normalized into a real value within the range of [0, 1]. The distance sensor returns a value that corresponds to the distance to the detected object or returns 0 if there are no objects within the sensor range.

The ground sensor can detect whether the robot is inside or outside a target area. The ground sensor returns a value of 1 if the robot is inside a target area and 0 otherwise.

In total, twenty input values are collected from the robot, i.e., twelve inputs from the omnidirectional camera, seven inputs from the distance sensors, and one input from the ground sensor. The input values are fed into the robot controller and return the output values for controlling the actuators. The robot actuators are controlled with four output values, i.e., two outputs for controlling the motors and another two for the activation of the front and rear LEDs. The output values take real values in the range [0, 1]. The outputs for the motors correspond to the rotation of the left and right wheels. The LEDs of the robot are turned on if the corresponding output value is higher than 0.5 or turned off otherwise.

4.2 Evolutionary Robotics Approach

The evolutionary robotics approach is a promising method for designing controllers of a robotic swarm. Typically, the evolutionary robotics approach designs controllers by evolving artificial neural networks [122], also known as neuroevolution [33]. In detail, an evolutionary algorithm evaluates and optimizes the robot controllers that are represented by neural networks. The evaluation of a controller is done based on a predefined fitness function, which indicates the achievement of the task addressed by the robotic swarm. The controllers with higher fitness values are selected to produce the next population of candidate controllers. The rest of this section describes the evolutionary robotics approach applied in this paper.

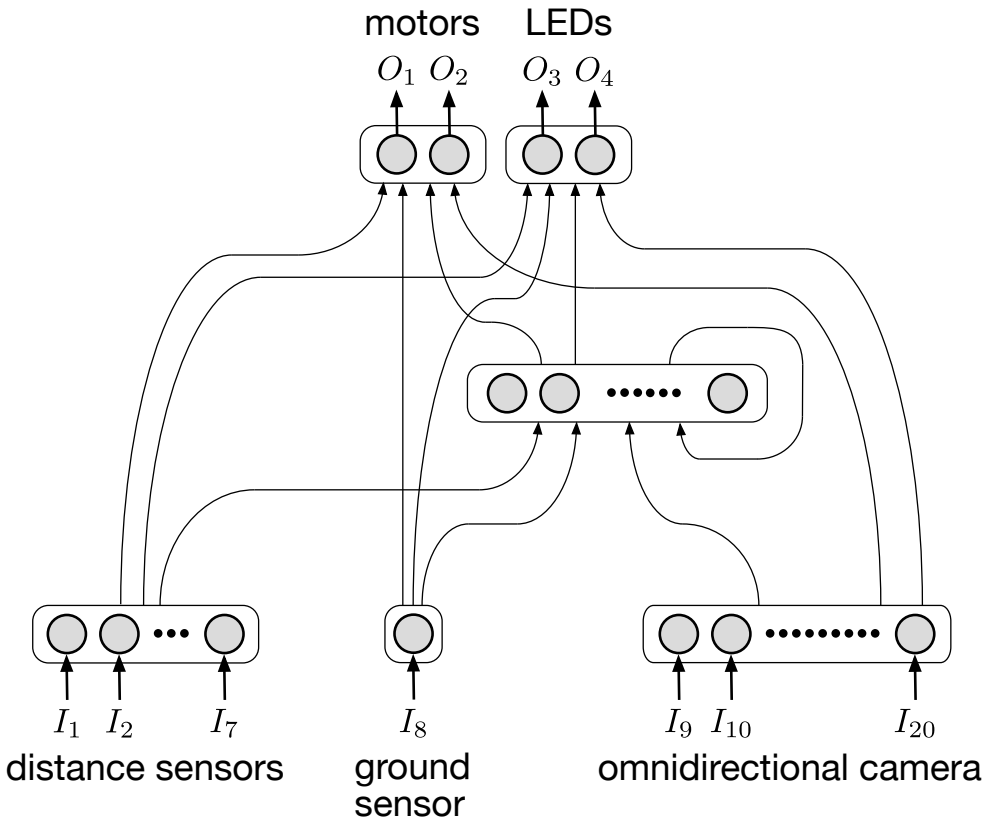


Fig. 4.3. Structure of the robot controller. The controller is represented by the recurrent neural network with ten hidden neurons.

4.2.1 Robot Controller

The controller of the robot is represented by a recurrent neural network, as shown in Fig. 4.3. The input layer is composed of twenty neurons; seven neurons from the distance sensors, one neuron from the ground sensor, and twelve neurons from the omnidirectional camera. The hidden layer is composed of ten neurons with recurrent connections, including self-connections. The output layer is composed of four neurons; two neurons for controlling the motors and two neurons for controlling the activation of the front and rear LEDs. The value of the kth neuron in the hidden layer $H_k(\tau)$ is updated with the following equations:

$$H_k(\tau) = \varphi_{\text{hid}}\left(\sum_i w_{ik}^{\text{IH}} I_i(\tau) + \sum_j w_{jk}^{\text{HH}} H_j(\tau - 1)\right), \tag{4.1}$$

$$\varphi_{\text{hid}}(x) = \frac{2}{1 + e^{-x}} - 1,$$

where $I_i(\tau)$ is the value from the ith neuron in the input layer at time τ, $H_j(\tau - 1)$ is the value from the jth neuron in the hidden layer at time $\tau - 1$, w_{ik}^{IH} is the synaptic weight from

the ith input neuron to the kth hidden neuron, and w_{jk}^{HH} is the synaptic weight from the jth hidden neuron to the kth hidden neuron. The value of the kth neuron in the output layer $O_k(\tau)$ is updated with the following equations:

$$O_k(\tau) = \varphi_{\mathrm{out}}\left(\sum_i w_{ik}^{\mathrm{IO}} I_i(\tau) + \sum_j w_{jk}^{\mathrm{HO}} H_j(\tau)\right), \tag{4.2}$$

$$\varphi_{\mathrm{out}}(x) = \frac{1}{1+e^{-x}},$$

where w_{ik}^{IO} is the synaptic weight from the ith input neuron to the kth output neuron, and w_{jk}^{HO} is the synaptic weight from the jth hidden neuron to the kth output neuron. Two different sigmoid activation functions φ_{hid} and φ_{out} are employed to scale the value of the hidden neuron H_k in the range $[-1, 1]$ and the value of the output neuron O_k in $[0, 1]$. All synaptic weights take real values in the range $[-1, 1]$. The robot activates the LEDs if the corresponding output neuron is larger than the threshold, i.e., turned on if the output value is higher than 0.5 and turned off otherwise. The output values for the motors control the rotation of the wheels. The function that converts the output values to the wheel rotations is estimated from the observation of the prototype physical robot. The values of neurons are updated every 13 simulation time steps, which are designed based on the processing speed of the physical robot.[*5]

4.2.2 Evolutionary Algorithm

The $(\mu,\ \lambda)$ evolution strategy [6, 7, 30] is employed for an evolutionary algorithm. The $(\mu,\ \lambda)$ evolution strategy is one of the classic evolutionary algorithms, which is often used to optimize real values. Table 4.1 shows the parameter settings of the $(\mu,\ \lambda)$ evolution strategy. The synaptic weights of the controller are optimized via the evolutionary algorithm. The evolutionary process lasts 1000 generations with the zeroth generation of a randomly generated population.

4.2.3 Fitness Function

The fitness function is defined based on the number of times the robots visit the two target areas alternately. A controller is copied to N robots and evaluated for $M = 3$ trials by executing the path-formation task with different random seeds. Each trial lasts for 7200 simulation time steps. During the first 1200 simulation time steps, the fitness value is not calculated, which is the period for the robots to move freely and explore the environment. Subsequently, the sub-fitness value $f_n(t)$, which is the fitness value for the nth robot at the

[*5] The time step is set using the recommended settings of the Box2D physics engine, i.e., a simulation time step equals 1/60 seconds. Hence, the neural network is calculated approximately every 0.2 s.

Table 4.1. Parameter settings of the (μ, λ) evolution strategy.

Parameter	Value
Number of parents μ	30
Number of offspring λ	200
Initial mutation step size	0.05
Range of the mutation step size	$[0.00001, 0.15]$

simulation time step t, is updated during the remaining 6000 simulation time steps by the following equation:

$$f_n(t) = f_n(t-1) + \begin{cases} 1 & \text{if the } n\text{th robot enters the different target area,} \\ 0 & \text{otherwise.} \end{cases} \tag{4.3}$$

This equation indicates that the sub-fitness value f_n will be incremented by 1 when the nth robot alternately visited the two target areas during 1200 and 7200 simulation time steps. The overall fitness value F, which is the fitness value for the controller, is calculated using the following equation:

$$F = \frac{1}{M}\sum_{m=1}^{M} F_m, \qquad F_m = \frac{1}{N}\sum_{n=1}^{N} f_n, \tag{4.4}$$

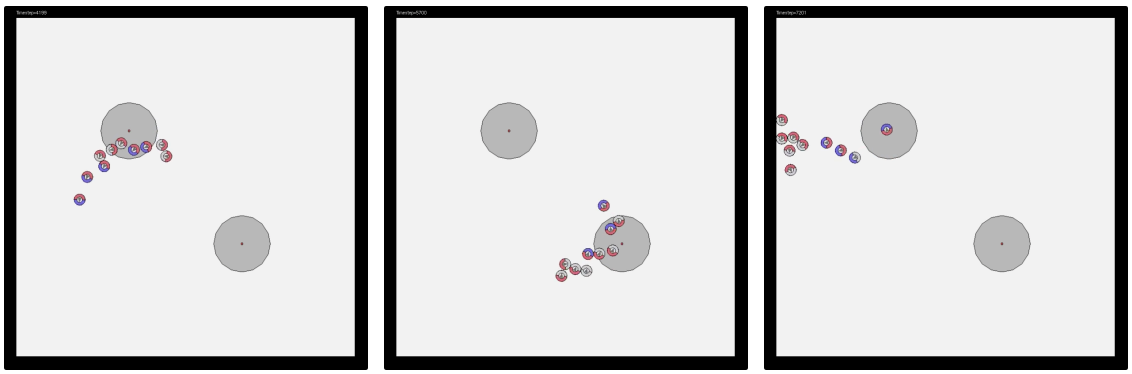
where M is the total number of trials, and F_m is the fitness of the mth trial, which is equal to the mean value of f_n over the number of robots N.

4.3 Experiments with Varying the Number of Robots

The path-formation task is performed with $N = 10$, 25, 50, 75, and 100 robots. For each setting of the simulation, five independent evolutionary processes are executed with a different random seed. At the end of each evolutionary process, the best controller within the last 100 generations is selected and re-evaluated for $M = 100$ trials. The set of weights that obtained the best fitness value in the re-evaluation is used for behavioral analysis.

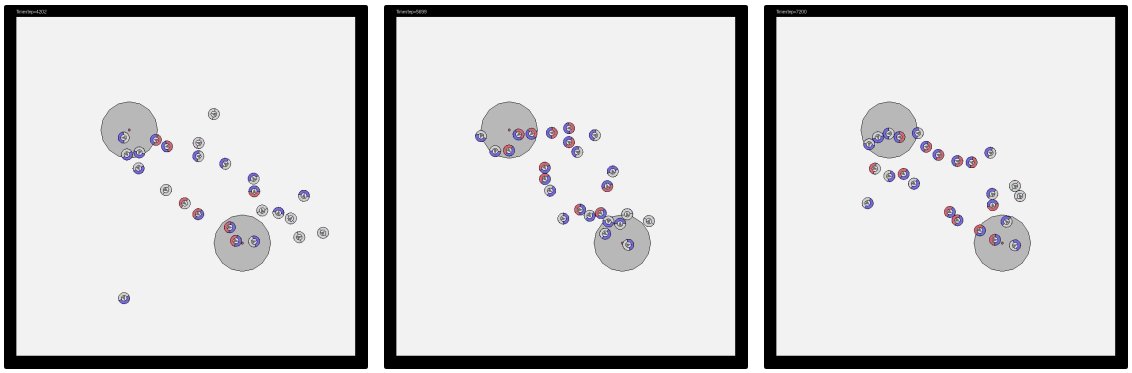
4.3.1 Results

The behaviors observed using the best-evolved controller are shown in Figs. 4.4 to 4.8. In addition, the results of the re-evaluation are shown in Fig. 4.9. The robot group with $N = 10$ failed to form a path due to the insufficient number of robots. Nevertheless, the swarm tended to coordinate its motion and explore collectively as a single group, as shown in Fig. 4.4. Indeed, the robots were able to form a path with $N = 25$ robots (see also Fig. 4.5). The fitness values scattered with $N = 25$ robots because the robotics swarm requires more



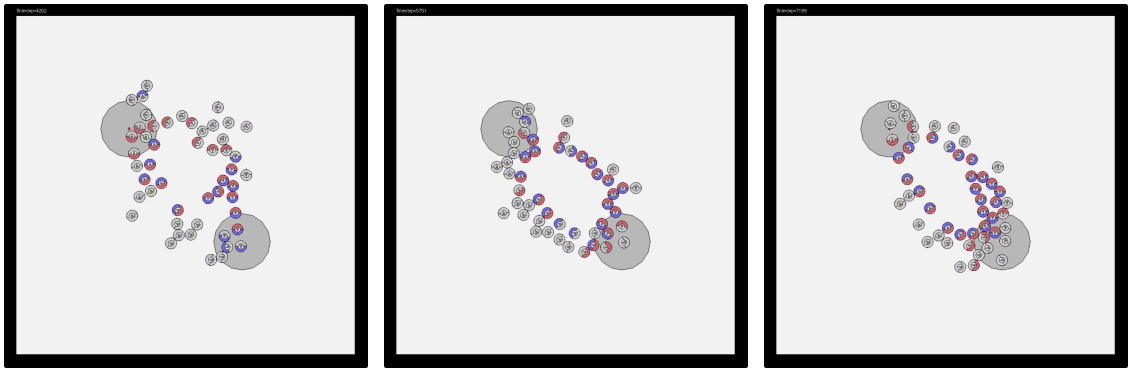
(a) 4200 time steps. **(b)** 5700 time steps. **(c)** 7200 time steps.

Fig. 4.4. Snapshots of the behavior observed using $N = 10$ robots.



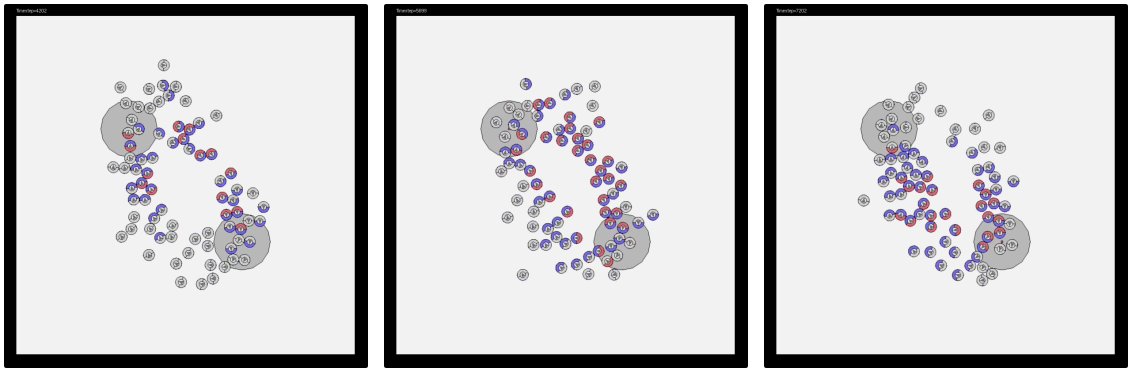
(a) 4200 time steps. **(b)** 5700 time steps. **(c)** 7200 time steps.

Fig. 4.5. Snapshots of the behavior observed using $N = 25$ robots.



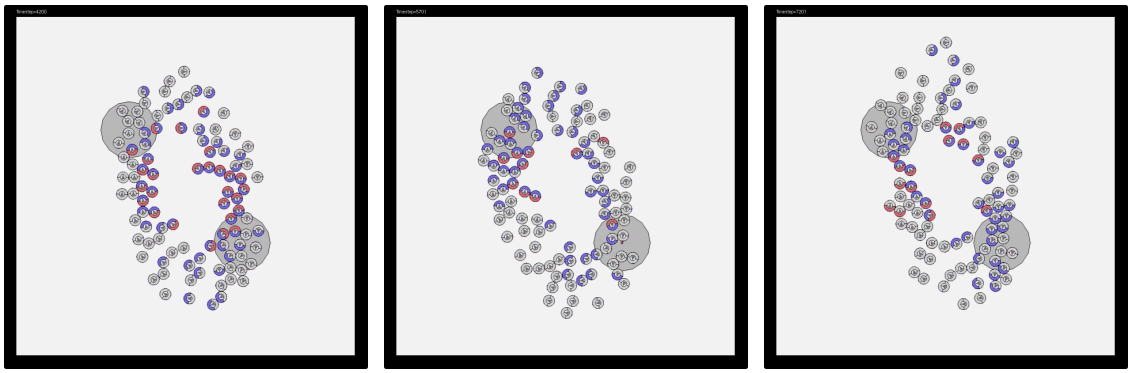
(a) 4200 time steps. **(b)** 5700 time steps. **(c)** 7200 time steps.

Fig. 4.6. Snapshots of the behavior observed using $N = 50$ robots.



(a) 4200 time steps. **(b)** 5700 time steps. **(c)** 7200 time steps.

Fig. 4.7. Snapshots of the behavior observed using $N = 75$ robots.



(a) 4200 time steps. **(b)** 5700 time steps. **(c)** 7200 time steps.

Fig. 4.8. Snapshots of the behavior observed using $N = 100$ robots.

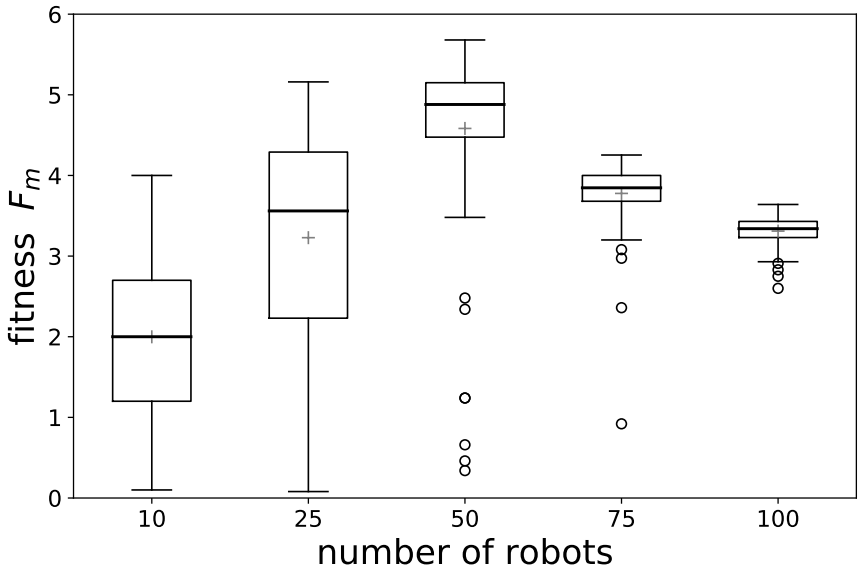


Fig. 4.9. Boxplots of the fitness values F_m using the best-evolved controller over $M = 100$ trials with varying the number of robots. Gray crosses mark the mean value.

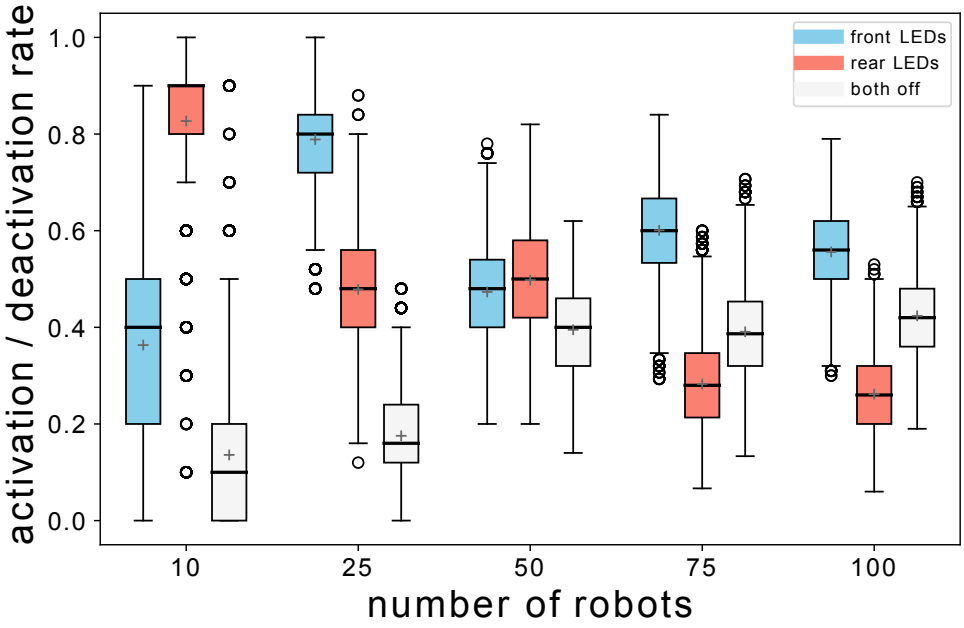


Fig. 4.10. Boxplots showing the percentage of robots activating or deactivating LEDs during 1200–7200 time steps. For each group of robots, the left boxplot refers to the activation rate of the front LEDs, the middle boxplot refers to the activation rate of the rear LEDs, and the rightbox plot refers to the rate with both LEDs deactivated.

time to aggregate, and in some trials, the robots failed to form a path. The highest fitness value was obtained with $N = 50$ robots, whereas the fitness values decreased as the number of robots increased. This tendency is because of congestion in forming a path. However, the total of landmark arrivals has been increased when discussed as a swarm (because the fitness F_m is averaged over the number of robots N). Simultaneously, the robotic swarm with larger group sizes was able to form a path much more stable across different trials.

It is worth noting that behavioral specialization appeared during the process of managing congestion. As can be seen from Figs. 4.5 to 4.8, as the number of robots increased, the robots inside the path activated their LEDs, while the other robots on the outer side deactivated them. The percentage of robots activating or deactivating LEDs during 1200 to 7200 time steps is shown in Fig. 4.10. The rate of robots deactivating both LEDs increased as the number of robots became larger. Approximately 10 to 15% of the robots on average deactivated both LEDs with $N = 10$, whereas 40% did so with $N = 100$. Furthermore, the rate of robots activating the rear LEDs decreased as the number of robots became larger. Approximately 30% of robots activated the rear LEDs with $N = 75$ and 100. It could be assumed that similar specialization behavior was obtained in $N = 75$ and 100 with having similar activation rates.

4.3.2 Discussion

For further analysis on the usage of LEDs, Fig. 4.11 shows the robot density in the environment and the trajectories of the robots activating or deactivating the LEDs with $N = 100$ robots. It could be said that the robotic swarm succeeded in generating a collective path because the two-dimensional histogram of robot density in Fig. 4.11(a) has a ring-shaped distribution. As can be seen from Fig. 4.11(b), the rear LEDs seem to be used as a guide to form the path, insofar as the robots traveling the innermost side of the path to activate them, whereas they are deactivated after the robots pass through the landmark. It seems that the robot decides to turn off the rear LEDs after arriving at a landmark not to confuse the other robots because the landmarks emit the same color as the rear LEDs. The rear LEDs are then reactivated after the robot travels some distance from the landmark. In addition, another behavioral specialization has emerged in the usage of the front LEDs. A comparison of Figs. 4.11(c) and (d) shows that the robots traveling along the inner side of the path activate their front LEDs, whereas those on the outer side deactivate them. The front LEDs are likely to be used to alert those robots traveling in the opposite direction. Thus, by these specializations, the robotic swarm was able to mitigate congestion and generate a path efficiently.

4.4 Evolutionary Acquisition of Behavioral Specialization in the Congested Path-formation Task

This section further discusses how the evolutionary robotics approach has developed the behavioral specialization in the robotic swarm. The experiments are performed by $N = 100$ robots so that the robotic swarm has to deal with the congestion problem. The results of evolutionary processes using $N = 100$ robots in Section 4.3 are selected for further discussion.

4.4.1 Results

As described in Section 4.3, five evolutionary processes were executed with a different random seed for designing the controller. The transitions of the highest fitness values of the five evolutionary processes are shown in Fig. 4.12. In all five evolutionary processes, the robotic swarm was able to generate a strategy to overcome the congestion problem.

The evolutionary process that obtained the highest fitness value among the five runs is selected for further analysis. As can be seen from Fig. 4.12, the fitness value rises steeply in the earlier stage of evolution. Therefore, the evolutionary process is divided the 1000 generations into two periods; the first 100 generations and the remaining 900 generations. Both periods are divided finely into a range of generations, i.e., the first period is divided into five ranges,

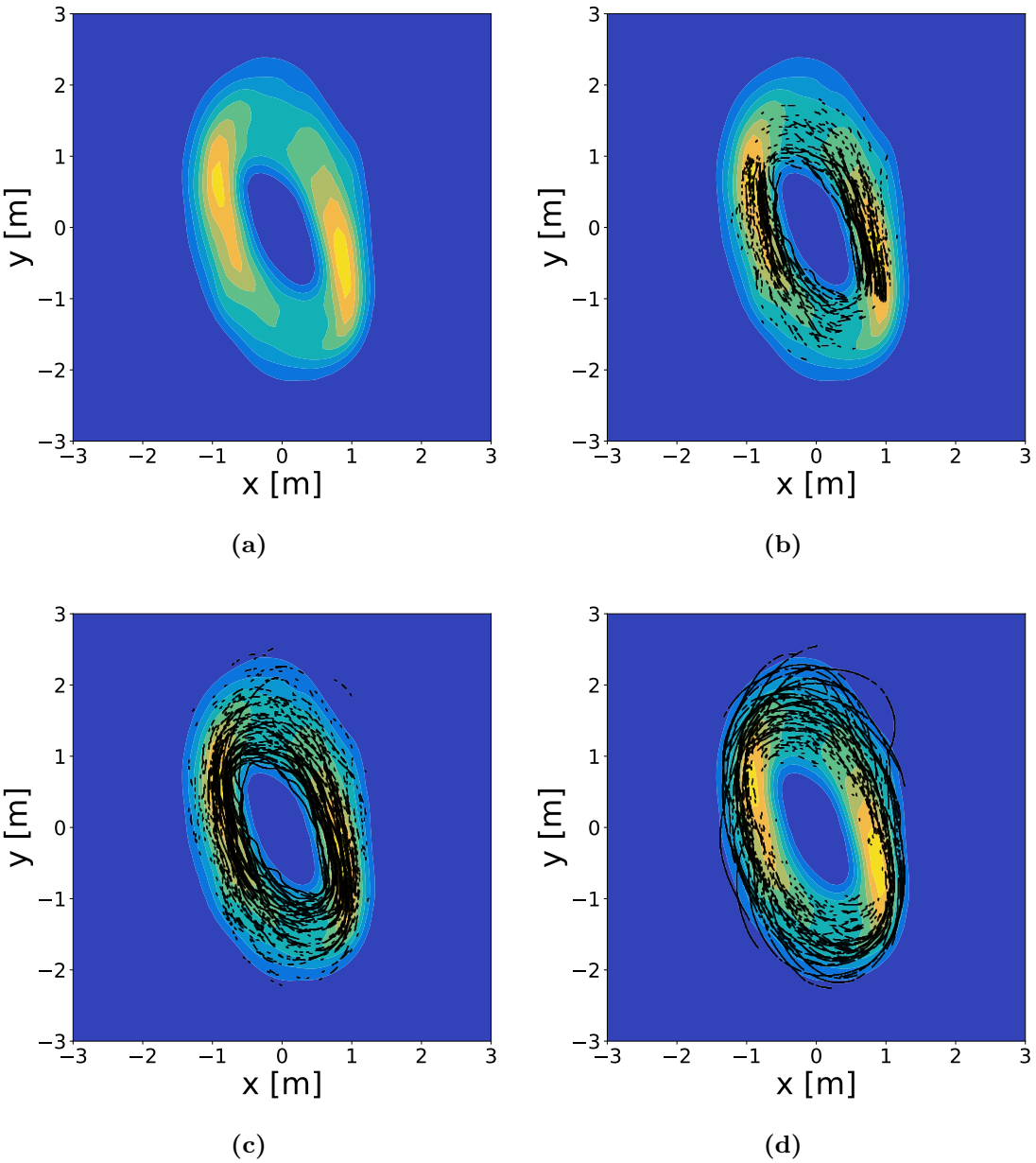


Fig. 4.11. (a) Two-dimensional histogram of the positions of robots within the environment during 6000–7200 time steps. The color gradient represents the robot density, i.e., the density increases from the dark color (blue) to the light color (yellow). Black lines and dots are plotted over the histogram if the robots are (b) activating the rear LEDs, (c) activating the front LEDs, and (d) deactivating both LEDs, during 6000 to 7200 time steps.

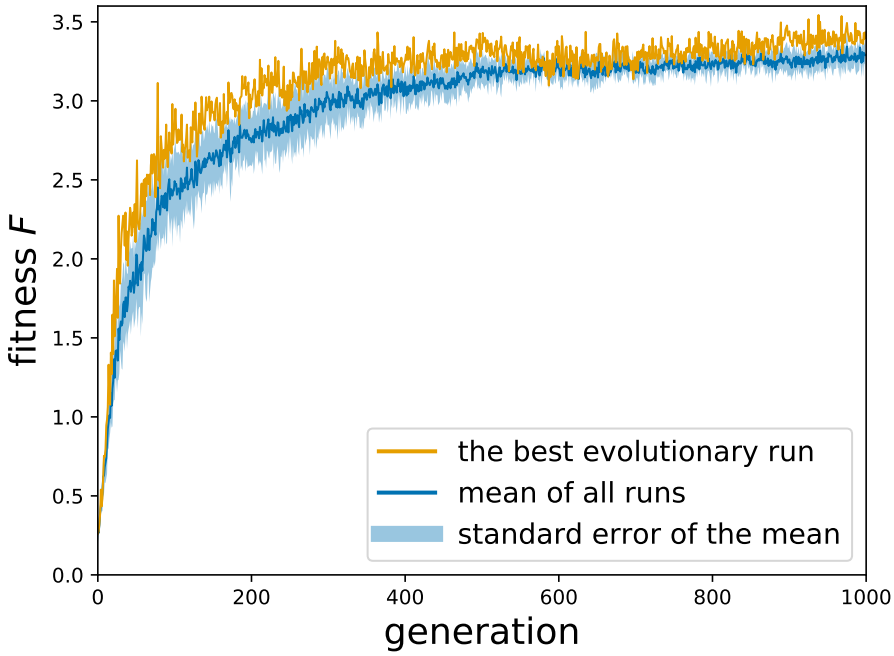


Fig. 4.12. Transitions of the highest fitness values of the five evolutionary runs. In the figure, "the best evolutionary run" indicates the transition of the evolutionary process that has obtained the highest fitness value among the five evolutionary processes.

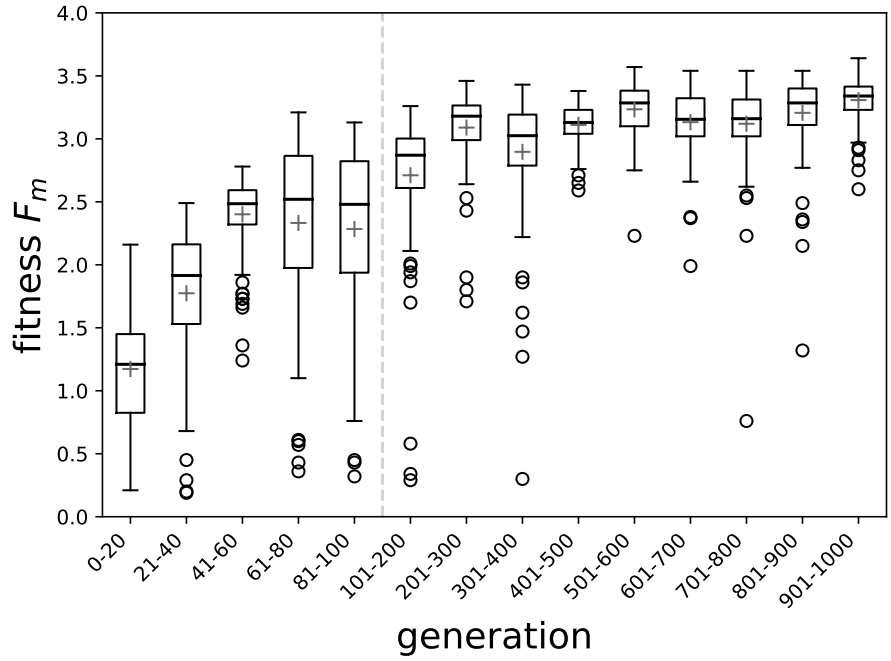
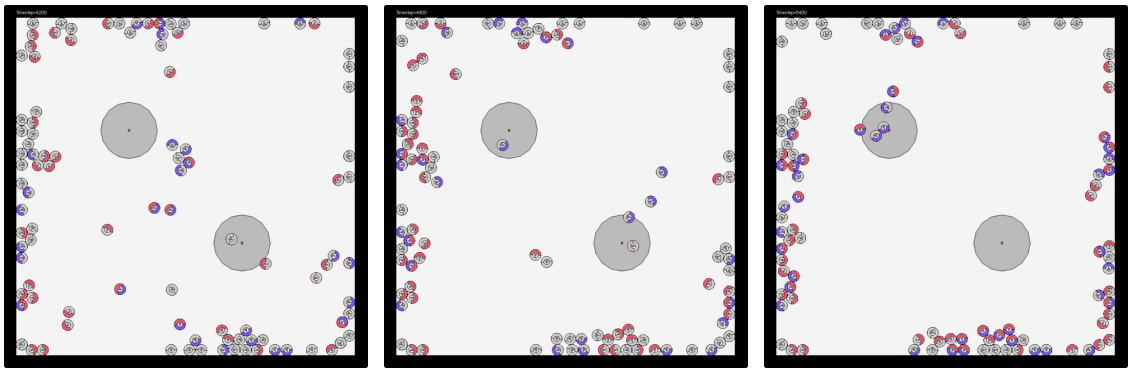
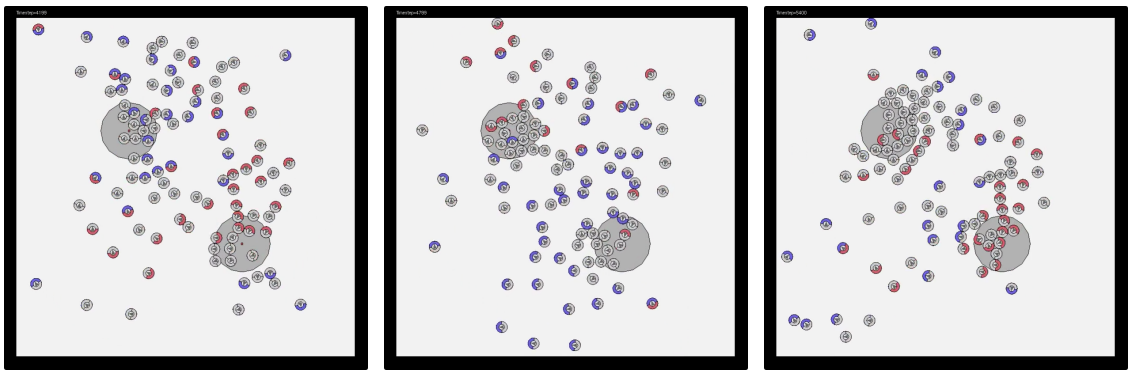


Fig. 4.13. Boxplots of the fitness values F_m using the best-evolved controller over $M = 100$ trials for each range of generations.



(a) 4200 time steps. **(b)** 4800 time steps. **(c)** 5400 time steps.

Fig. 4.14. Snapshots of the behavior observed using the best controller in the zeroth generation.

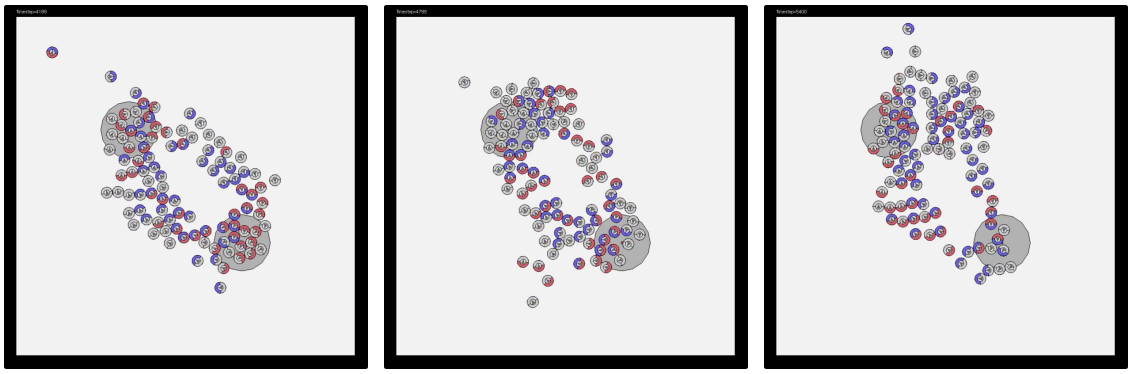


(a) 4200 time steps. **(b)** 4800 time steps. **(c)** 5400 time steps.

Fig. 4.15. Snapshots of the behavior observed using the best controller in the generation range 0–20.

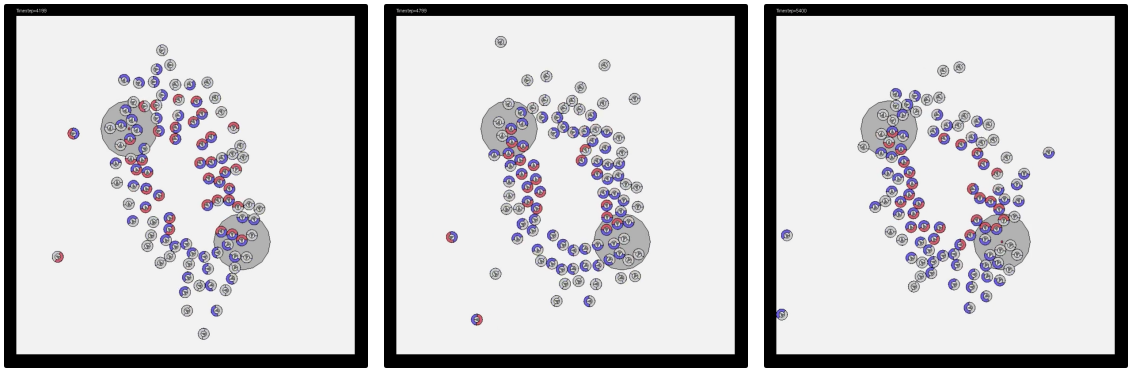
and the later period is divided into nine ranges. The controller that obtained the highest fitness in each range of generations is selected and re-evaluated for $M = 100$ trials. The results of the re-evaluation using the best-evolved controller in each range of generations are shown in Fig. 4.13. In the later stage of the evolution, the robotic swarm obtained higher fitness values and was able to form a path more stable.

The behaviors observed using the best-evolved controller within 0 to 20, 101 to 200, 401 to 500, and 901 to 100 generations are shown in Figs. 4.15, 4.16, 4.17, and 4.18, respectively. In addition, the behavior observed using the controller in the zeroth generation, which uses the randomly generated synaptic weights for the controller, is shown in Fig. 4.14. In the zeroth generation, almost all robots collide with each other and with walls (see also Fig. 4.14). Within 20 generations, although the robotic swarm does not form a collective path, a swarm-level behavior to travel clockwise has already emerged, as shown in Fig. 4.15.



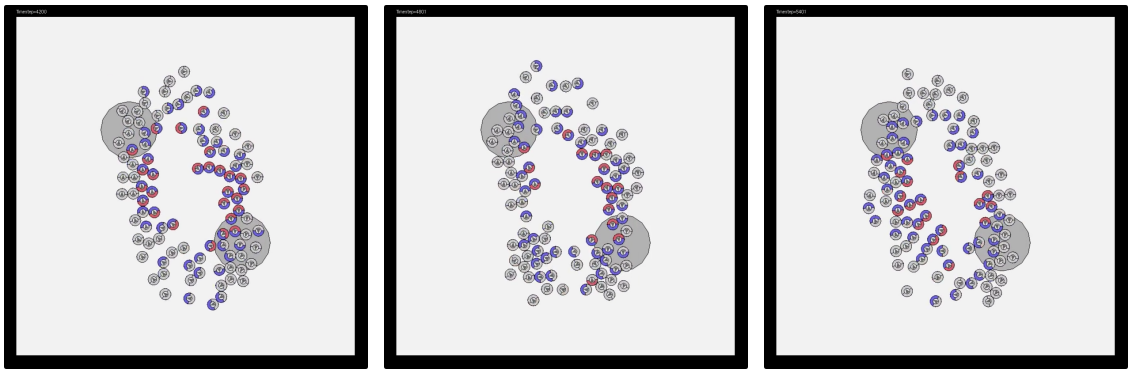
(a) 4200 time steps. **(b)** 4800 time steps. **(c)** 5400 time steps.

Fig. 4.16. Snapshots of the behavior observed using the best controller in the generation range 101–200.



(a) 4200 time steps. **(b)** 4800 time steps. **(c)** 5400 time steps.

Fig. 4.17. Snapshots of the behavior observed using the best controller in the generation range 401–500.



(a) 4200 time steps. **(b)** 4800 time steps. **(c)** 5400 time steps.

Fig. 4.18. Snapshots of the behavior observed using the best controller in the generation range 901–1000.

The robotic swarm was relatively stable at forming a path in generation range 101 to 200; however, the congestion has occurred near the landmarks, as shown in Figs. 4.16(b) and (c). To manage the congestion, the robotic swarm emerges a behavioral specialization, as shown in Figs. 4.17 and 4.18, the robots traveling inside the path activate the LEDs, while the robots at the outer side deactivate them.

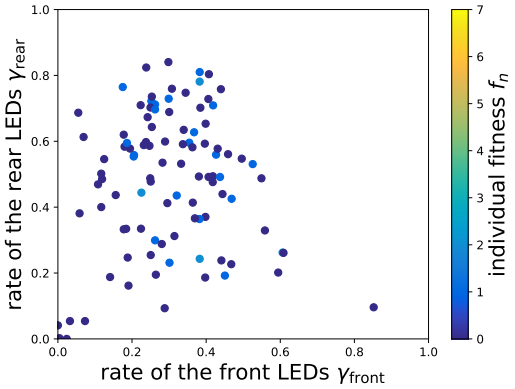
4.4.2 Discussion

The activation rates of the LEDs during the robots traveling between the landmarks are calculated to discuss the behavioral specialization that emerged in the robotic swarm. For each robot, the activation rate of the LEDs during 1200 to 7200 time steps is calculated by the following equation:

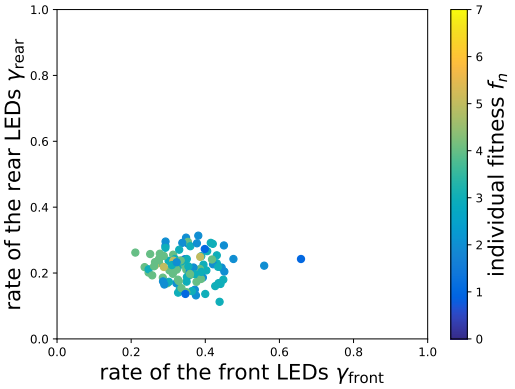
$$\gamma_{\text{front/rear}} = \frac{\tau_{\text{front/rear}}}{T}, \tag{4.5}$$

where $\gamma_{\text{front/rear}}$ is the activation rate of front or rear LEDs, $\tau_{\text{front/rear}}$ is the total time steps of the robot activating the front/rear LEDs during 1200 to 7200 time steps, and T $(= 7200 - 1200 = 6000)$ is the total time steps of calculating the rates. The activation rates for 100 robots in the observed behavior of Figs. 4.14 to 4.18 are shown in Fig. 4.19. In Fig. 4.19, the scatter plots show the front versus rear activation rates. In addition to the activation rates, Fig. 4.19 shows the sub-fitness value f_n in Eq. (4.3), which indicates the number of times the corresponding robot entered the landmarks during 1200 to 7200 time steps.

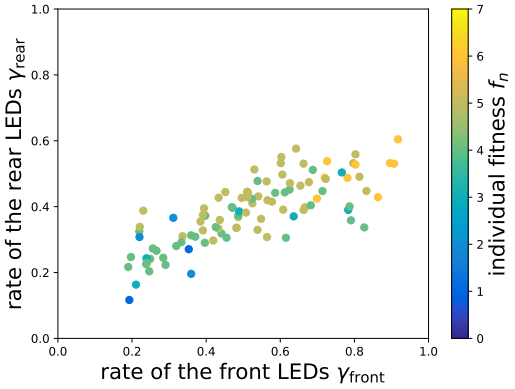
In the zeroth generation, there is no clear tendency in the activation rates and the sub-fitness value f_n, as shown in Fig. 4.19(a). Within 20 generations, the distribution of the activation rates seems to be centered at approximately $\gamma_{\text{front}} = 0.4$ and $\gamma_{\text{rear}} = 0.2$ (see also Fig. 4.19(b)). In the early stage of evolution, almost all robots have similar activation rates, which indicates that all robots behave similarly to each other. In other words, there is no specialized individual in an earlier stage of evolution. After a few progress of evolution, the activation rate of the front LEDs γ_{front} was distributed widely across 0.1 to 0.9, as shown in Fig. 4.19(c). In this middle stage of evolution, the robots learned to follow red LEDs and avoid blue LEDs to form a path. However, the rules of the usage of red LEDs have not yet emerged in this stage of evolution. The robots seem to get confused with detecting the red LEDs of the landmark and robots, which leads to congestion near landmarks. In a later stage of evolution, the distribution of the activation rates is in an orderly pattern of a curved line, as shown in Figs 4.19(d) and (e). Considering behavior observed in Figs. 4.17 and 4.18, the robots learned to specialize their behavior to perform their respective roles, viz., a wide range of roles in the activation of LEDs from activating almost always to almost none. Furthermore, there is a tendency that the higher the activation rates of LEDs, the higher the sub-fitness f_n. These results indicate that the robots traveling the more inner



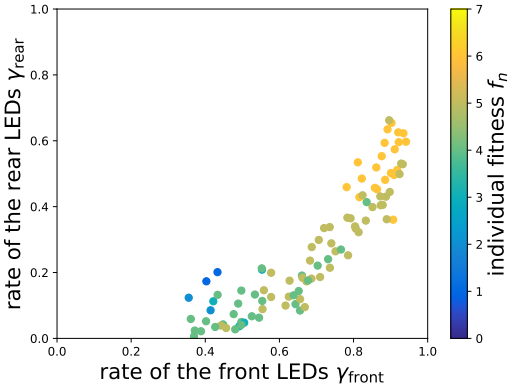
(a) Zeroth generation.



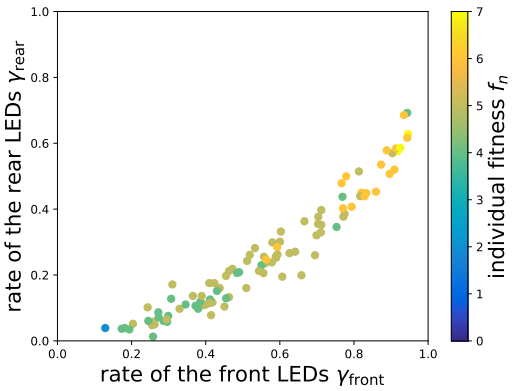
(b) Generation range 0–20.



(c) Generation range 101–200.



(d) Generation range 401–500.



(e) Generation range 901–1000.

Fig. 4.19. Scatter plots of the activation rates of the LEDs during 1200–7200 time steps. Each point indicates the robot with corresponding LED activation rates. The color of the point shows the sub-fitness value f_n of the corresponding robot.

part of the path had higher activation rates. In contrast, the robot with the lower activation rates marks lower sub-fitness f_n, which implies that the robots traveling the outer part of the path deactivated the LEDs. Thus, by these specializations, the robotic swarm was able to manage congestion.

4.5 Conclusions

This chapter developed a controller for a robotic swarm to address a path-formation task by applying the evolutionary robotics approach. The experiments were conducted by varying the number of robots. The robots specialized their behavior and exhibited task allocation to manage the congestion that occurred in larger swarm sizes. The robotic swarm performed specialization in that the robots traveling inside the path activated the LEDs while the robots on the outer side deactivated them. The strategy to manage congestion was evolved from scratch without providing any prior knowledge regarding how to accomplish the task.

Chapter 5

Systematic Investigation of Behavioral Specialization: Effects of Congestion and Embodiment

This chapter further discusses the behavioral specialization developed in Chapter 4. The studies on swarm robotics emphasize the importance of the embodiment of robots. However, there have been only a few studies on how the embodiment influences the collective behavior of robotic swarms. For example, there have been studies on the relationship between swarm performance and the number of robots [47, 48, 75]. These studies have shown that an excessive number of robots leads to interference among robots, which decreases the performance of the individual robot. Besides, most of these studies discussed swarm performance in terms of task accomplishment (e.g., the time required to complete a task), but only a few discussions on emergent collective behavior to solve a task.

This chapter discusses the effect of the robot embodiment on swarm performance and behavioral specialization from two aspects. First, the effects of congestion on swarm performance are discussed by considering the size of robots. The experiments are conducted by varying the number of robots and the robot size to change the degree of congestion. Next, how collisions among the robots affect the collective behavior of robotic swarms are discussed by conducting computer simulations with and without considering collisions among robots. These experiments show that the embodiment of the robots is also an essential feature to discuss swarm performance and specialization.

The rest of this chapter is organized as follows. Section 5.1 describes the settings of the experiments. Section 5.2 presents and discusses the results of the experiments conducted by varying the number of robots and the robot size. The results of the experiments conducted with and without considering collisions among robots are presented and discussed

Table 5.1. Experiment settings of the path-formation task with varying the number of robots and the robot size.

Parameter	Value
Number of robots	10, 25, 50, 75, 100
Diameter of the robot	0.1, 0.2, 0.4 [m]

in Section 5.3. Finally, Section 5.4 concludes this chapter.

5.1 Settings of the Experiments

In this chapter, the same experiment settings are employed as described in Chapter 4. The details of the settings of the path-formation task are described in Section 4.1. The settings of the evolutionary robotics approach are described in Section 4.2.

5.2 Effects of Congestion on Swarm Performance and Behavioral Specialization

This section focuses on the effect of congestion on the swarm performance by considering the number of robots and the robot size. Therefore, the experiments are conducted with the settings described in Table 5.1. The path formation is performed with $N = 10$, 25, 50, 75, and 100 robots. As for the robot size, the diameter of 0.2 m is set as the standard robot size, which is the size described in Chapter 4. In addition, the settings with half and twice the diameter are employed to consider the effect of the robot size. Five independent evolutionary processes are executed for each pair of settings with a different random seed. At the end of the evolutionary process, the synaptic weights that obtained the highest fitness value within the last 100 generations are selected and re-evaluated for $M = 100$ trials. The best synaptic weights in this re-evaluation are used for further analysis.

5.2.1 Results

The results of the re-evaluation using the best synaptic weights are shown in Fig. 5.1. In the experiments with $N = 10$ robots, although the robot diameter of 0.1 m has slightly lower performance, the robotic swarms have scored relatively similar fitnesses which distribute within the range $[0, 4]$. These results are because $N = 10$ robots are insufficient to form a path regardless of the robot size.

First, the performance is compared with the robot diameter of 0.1 m and 0.2 m to examine the effect of decreasing the robot size. There was a significant difference in the performance of robotic swarms with $N = 10$ and 25 robots (Mann-Whitney U test, $N = 10$ robots,

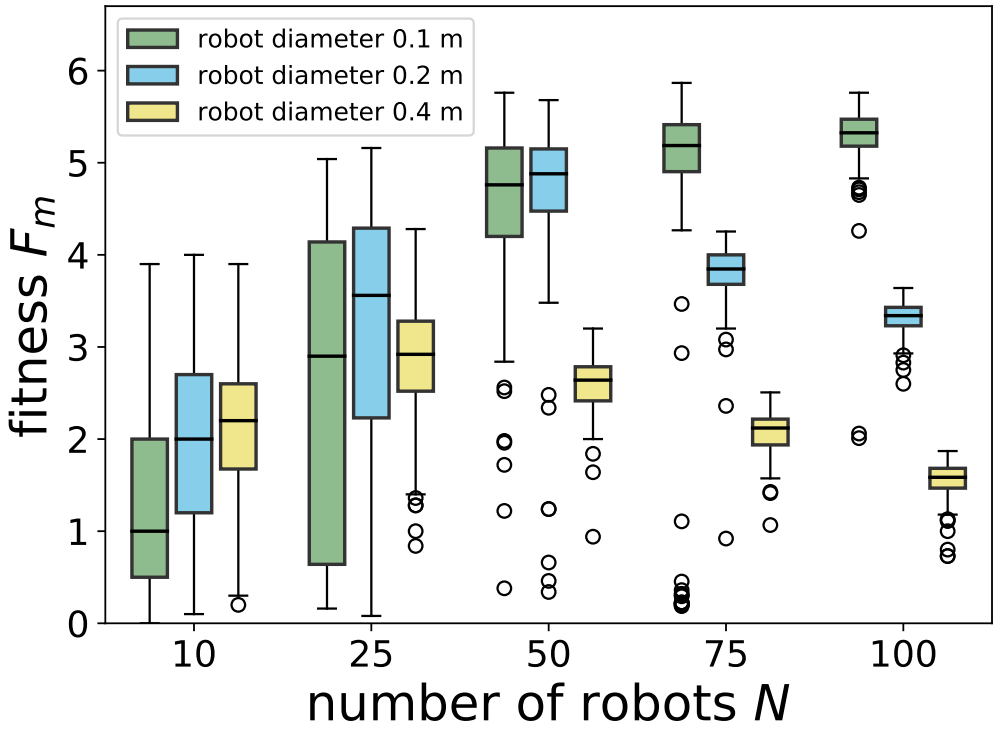


Fig. 5.1. Boxplots of the fitness values F_m using the best-evolved weights over $M = 100$ trials using different sized robots. For each number of robot N, the left (green) box plot refers to the robot size with the diameter of 0.1 m, the middle (blue) box plot refers to 0.2 m, and the right (yellow) box plot refers to 0.4 m.

p-value < 0.001; $N = 25$ robots, p-value < 0.05). However, in both cases, the fitness values were distributed over similar ranges; i.e., a range of $[0, 4]$ with $N = 10$ robots and a range of $[0, 5]$ with $N = 25$ robots. There was no significant difference between the performance with the diameter of 0.1 m and 0.2 m in $N = 50$ robots (p-value > 0.05). On the other hand, in cases with $N = 75$ and 100 robots, the performance with the 0.1 m diameter was significantly higher than those of 0.2 m diameter (p-value < 0.001). These results are because there is less congestion in situations with smaller robot sizes.

Next, the performance is compared with the robot diameter of 0.2 m and 0.4 m. When comparing the robotic swarm with the diameter of 0.2 m and 0.4 m, there was no significant difference between the performance with $N = 10$ robots (p-value > 0.05). In cases with $N = 25$, 50, 75, and 100 robots, the performance with the 0.2 m diameter was significantly higher than those of 0.4 m diameter (p-value < 0.001). These results are because congestion is more likely to occur in the larger robot size. Moreover, with larger-sized robots, fewer robots can enter the target area at the same time.

In experiments with the 0.2 m robot diameter, which are the settings with the standard

size described in Section 4.3, the highest fitness value was scored with $N = 50$ robots. The examples of behavior observed using $N = 25$, 50, and 100 robots are shown in Figs. 5.2, 5.3, and 5.4, respectively. When N is higher than 50 robots, the robotic swarm emerges behavioral specialization; i.e., the robots traveling the inner side of the path activate the LEDs, while those in the outer side deactivate them. This specialization becomes more pronounced when the number of robots increases (see also Figs. 5.3 and 5.4). More details on the behavior that was observed using the robots with the 0.2 m diameter were described in Chapter 4.

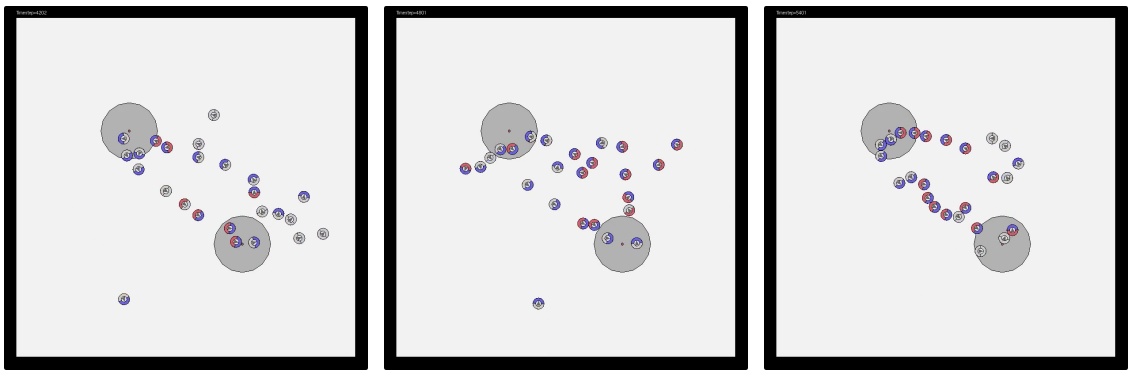
In the experiments using the 0.1 m diameter, the fitness values increased as the number of robots became larger and plateaued around 5.5, as shown in Fig. 5.1. It could be assumed that there is less congestion in the robotic swarm when using the 0.1 m diameter robots. The examples of behavior observed using $N = 25$, 50, and 100 robots are shown in Figs. 5.5, 5.6, and 5.7, respectively. Similar to the behavior with the 0.2 m diameter robots, the robotic swarm emerged the behavioral specialization in the robotic swarm with a larger number of robots.

As for the robots with the 0.4 m diameter, the highest fitness value was scored with $N = 25$ robots. The examples of behavior observed using $N = 25$, 50, and 100 robots are shown in Figs. 5.8, 5.9, and 5.10, respectively. The robotic swarm with $N = 50$ robots emerged a similar behavioral specialization that was observed in 0.2 m diameter robots (see also Fig. 5.9). On the other hand, this specialization is no longer observable when the number of robots is increased (see also Fig. 5.10). Further discussion of the behavioral specialization and its analysis is in the next section.

5.2.2 Discussion

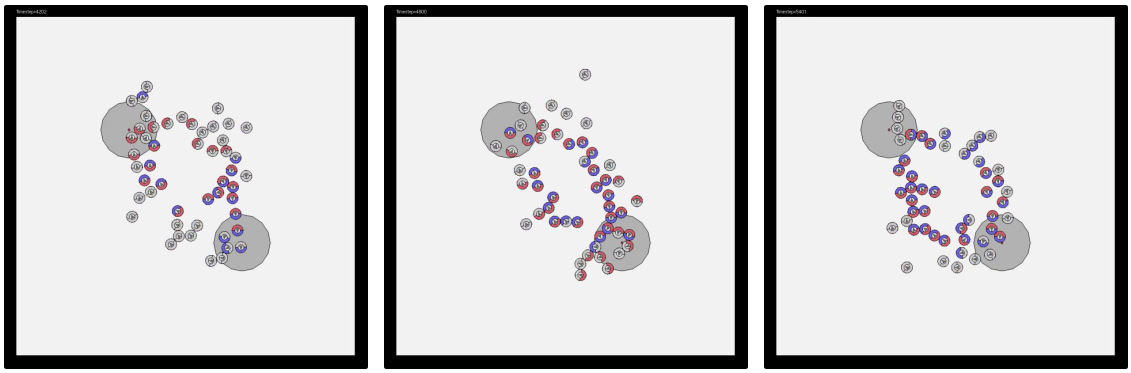
The activation rates of the LEDs are calculated to discuss the behavior of the robotic swarm. For each robot, the activation rates of the LEDs during 1200 to 7200 time steps are calculated by Eq. (4.5). The activation rate indicates the percentage of time steps that the robot activated the corresponding LEDs during 1200 to 7200 time steps. The activation rates are calculated based on the behavior observed with $N = 25$, 50, and 100 robots. The scatter plots of the front versus rear LED activation rates are shown in Figs. 5.11, 5.12, and 5.13. In addition to the activation rates, the colors in Figs. 5.11, 5.12, and 5.13 show the sub-fitness value f_n in Eq. (4.3) during 1200 to 7200 time steps. The sub-fitness value f_n indicates the number of times the corresponding robot entered the landmarks.

The results using robots with the 0.2 m diameter robots show that the specialization emerged as the number of robots increased. As can be seen from Fig. 5.11(a), the distribution of the activation rates is centered around $\gamma_{\text{front}} = 0.8$ and $\gamma_{\text{rear}} = 0.5$. This result implies that all robots have a similar strategy on activating LEDs, i.e., the specialization has not emerged



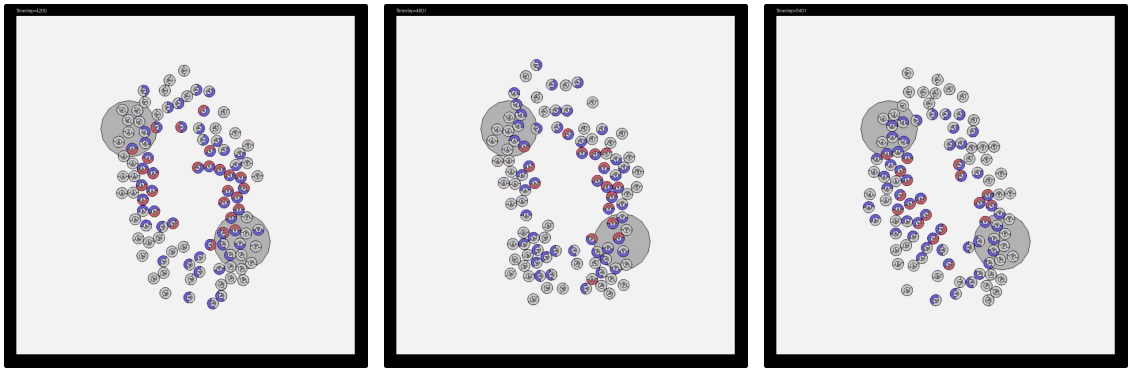
(a) 4200 time steps. **(b)** 4800 time steps. **(c)** 5400 time steps.

Fig. 5.2. Snapshots of behavior observed using 25 robots with the diameter of 0.2 m.



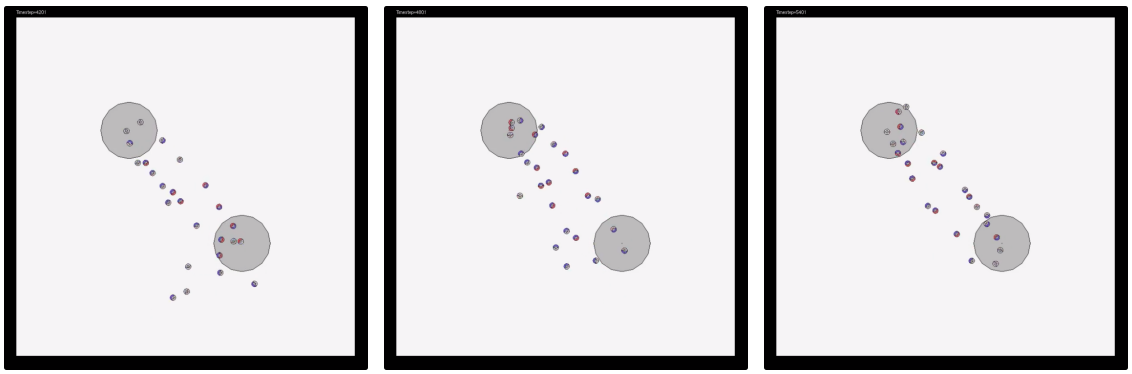
(a) 4200 time steps. **(b)** 4800 time steps. **(c)** 5400 time steps.

Fig. 5.3. Snapshots of behavior observed using 50 robots with the diameter of 0.2 m.



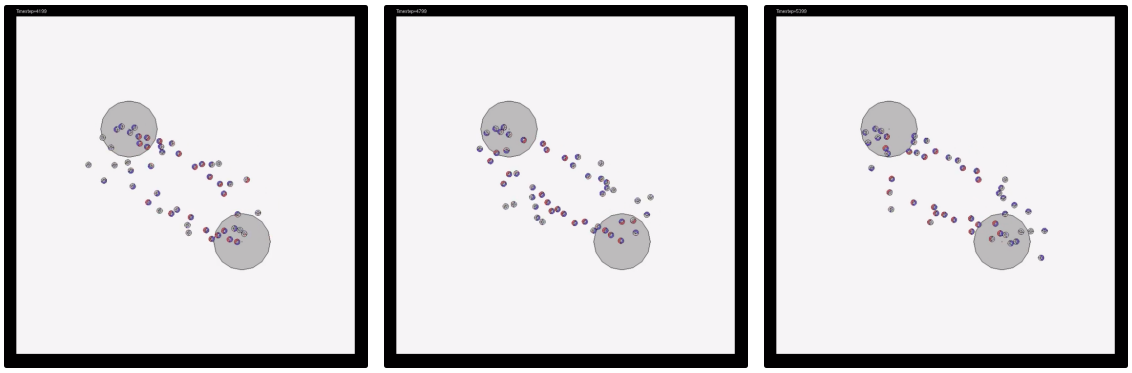
(a) 4200 time steps. **(b)** 4800 time steps. **(c)** 5400 time steps.

Fig. 5.4. Snapshots of behavior observed using 100 robots with the diameter of 0.2 m.



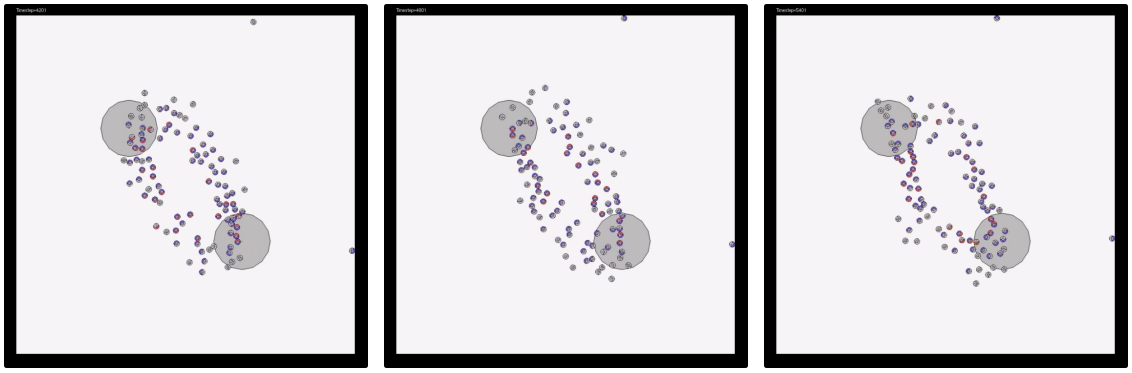
(a) 4200 time steps. **(b)** 4800 time steps. **(c)** 5400 time steps.

Fig. 5.5. Snapshots of behavior observed using 25 robots with the diameter of 0.1 m.



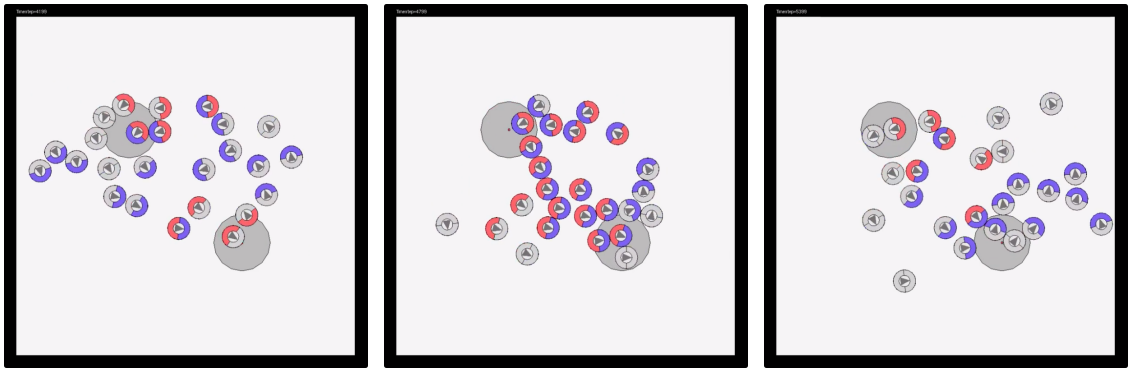
(a) 4200 time steps. **(b)** 4800 time steps. **(c)** 5400 time steps.

Fig. 5.6. Snapshots of behavior observed using 50 robots with the diameter of 0.1 m.



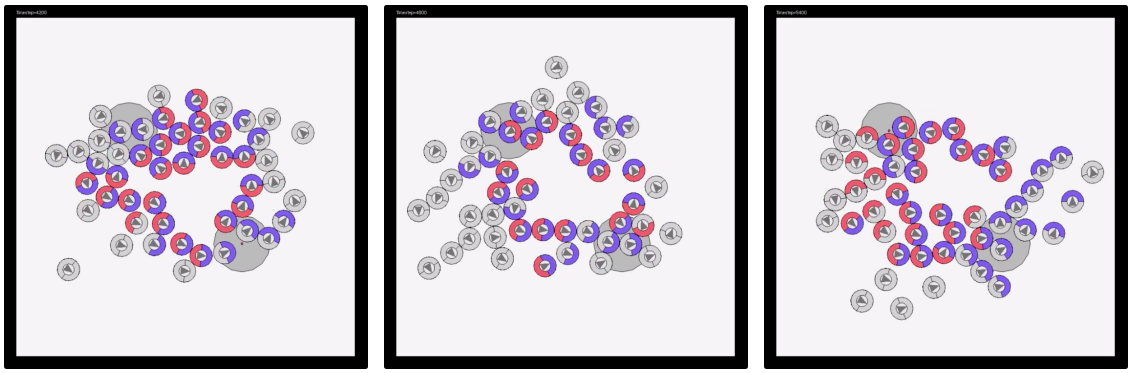
(a) 4200 time steps. **(b)** 4800 time steps. **(c)** 5400 time steps.

Fig. 5.7. Snapshots of behavior observed using 100 robots with the diameter of 0.1 m.



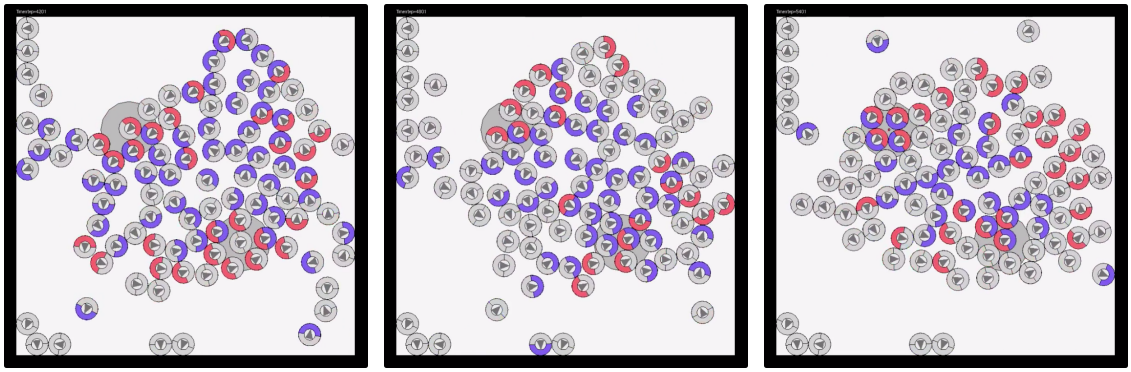
(a) 4200 time steps. **(b)** 4800 time steps. **(c)** 5400 time steps.

Fig. 5.8. Snapshots of behavior observed using 25 robots with the diameter of 0.4 m.



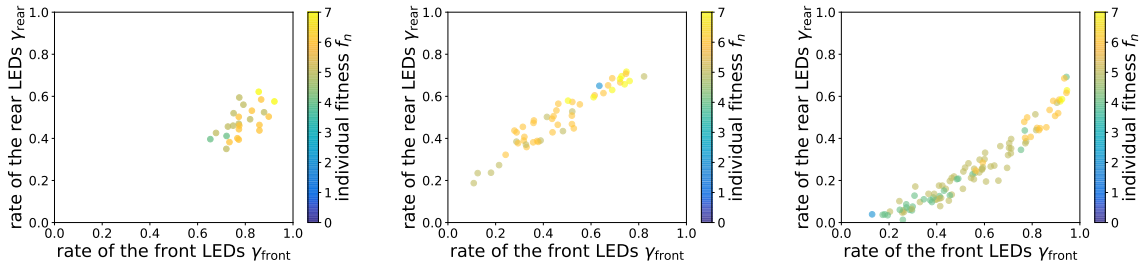
(a) 4200 time steps. **(b)** 4800 time steps. **(c)** 5400 time steps.

Fig. 5.9. Snapshots of behavior observed using 50 robots with the diameter of 0.4 m.



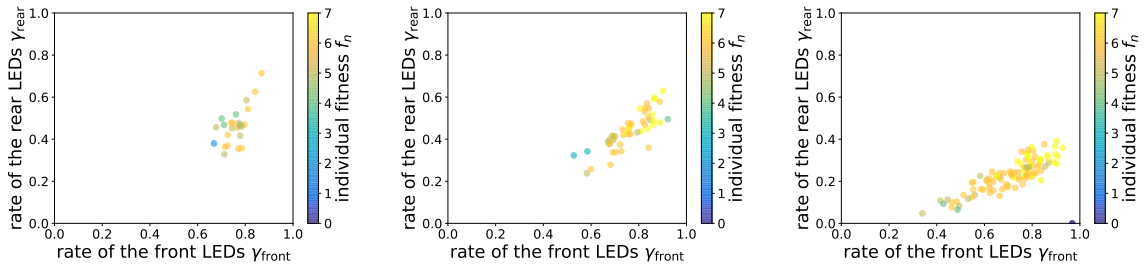
(a) 4200 time steps. **(b)** 4800 time steps. **(c)** 5400 time steps.

Fig. 5.10. Snapshots of behavior observed using 100 robots with the diameter of 0.4 m.



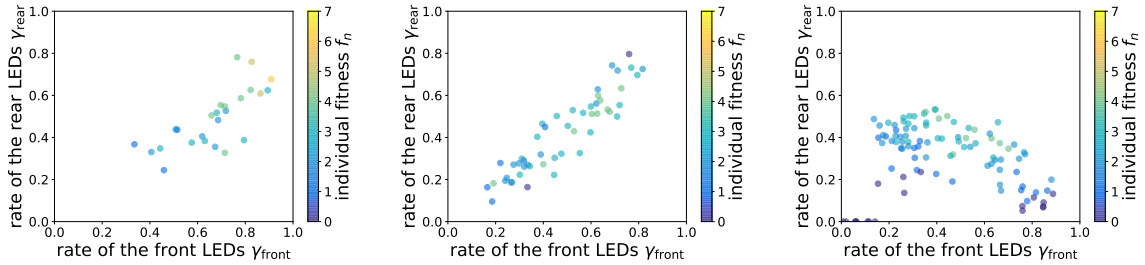
(a) 25 robots. **(b)** 50 robots. **(c)** 100 robots.

Fig. 5.11. Scatter plots of the activation rate of the LEDs during 1200–7200 time steps with the robot diameter of 0.2 m. Each point indicates the robot with corresponding activation rates. The color of the point shows the sub-fitness value f_n of the corresponding robot.



(a) 25 robots. **(b)** 50 robots. **(c)** 100 robots.

Fig. 5.12. Scatter plots of the activation rate of the LEDs during 1200–7200 time steps with the robot diameter of 0.1 m.



(a) 25 robots. **(b)** 50 robots. **(c)** 100 robots.

Fig. 5.13. Scatter plots of the activation rate of the LEDs during 1200–7200 time steps with the robot diameter of 0.4 m.

in 25 robots. The distribution of the activation rates spreads with an increase in the number of robots and shows a positive correlation between the two activation rates, as can be seen from Figs. 5.11(b) and (c). Moreover, almost all robots obtained relatively high sub-fitness f_n in 50 robots, as shown in Fig. 5.11(b). In 100 robots, however, the robots with lower activation rates have lower sub-fitness f_n, as shown in Fig. 5.11(c). The robots traveling the outer side of the path are more inefficient to perform the task, and therefore, they obtained a lower sub-fitness f_n. Hence, Fig. 5.11(c) shows that more robots were traveling the outer side of the path and deactivating the LEDs in the robotic swarm of 100 robots. These results show that specialization becomes more pronounced as the number of robots increases.

The results with the 0.1 m diameter show a similar tendency with the diameter of 0.2 m. In 25 robots, the activation rates tend to aggregate near $\gamma_{\text{front}} = 0.8$ and $\gamma_{\text{rear}} = 0.5$, as can be seen from Fig. 5.11(a). As the number of robots increases, the distribution of the activation rates shows a positive correlation, as can be seen from Figs. 5.11(b) and (c). However, the distribution of the activation rates seems to have less dispersion and obtained higher sub-fitness f_n than that of the 0.2 m diameter. It seems that the robotic swarm with the 0.1 m diameter robots has weaker specialization as compared to the diameter of 0.2 m.

In contrast, the robots with 0.4 m diameter have lower sub-fitness f_n because of the congestion, as shown in Fig. 5.13. The results using 25 robots show that the distribution of the activation rates with the diameter of 0.4 m is more scattered than that of 0.2 m (see also Figs. 5.11(a) and 5.13(a)). These results indicate that the specialization has already started to emerge in 25 robots with the 0.4 m diameter robots. There is also a positive correlation between the two activation rates in 50 robots, as shown in Fig. 5.11(b); however, robots with higher activation rates scored lower sub-fitness f_n. This result is because of the congestion near the landmarks; i.e., the robots traveling inside the path were pushed farther inward, and consequently, those robots failed to enter the target area. In the swarm of 100 robots, the positive correlation between the two activation rates disappeared, as shown in Fig. 5.13(c). Thus, it can be concluded that the robot size is an essential feature when discussing the relationship between specialization and congestion.

5.3 Effects of the Robot Embodiment on Behavioral Specialization

This section focuses on the effect of collisions among robots on the collective behavior of robotic swarms. The path-formation task is performed with $N = 100$ robots, with and without considering robot collisions. Additionally, the experiments are performed with the robots with a diameter of 0.1, 0.2, and 0.4 m. The robots with a larger size are more likely to collide with each other. In settings without considering collisions, the robots may overlap

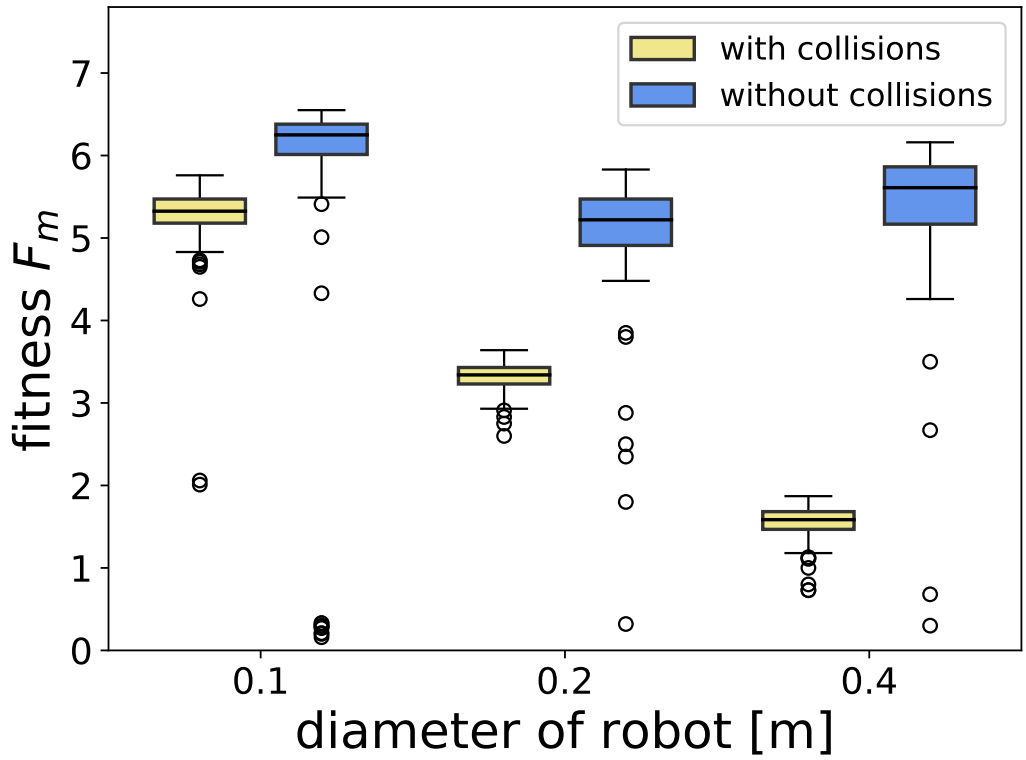


Fig. 5.14. Box plots of the fitness F_m over $M = 100$ trials for experiment settings with and without considering the robot collisions.

and pass through each other without colliding.[*6] However, their LEDs could be detected by the omnidirectional camera equivalent to the settings with collisions. Five independent evolutionary processes are executed for each experiment settings with a different random seed. At the end of the evolutionary process, the synaptic weights that scored the highest fitness value within the last 100 generations are selected and re-evaluated for $M = 100$ trials.

5.3.1 Results

The results of the re-evaluation using the best synaptic weights are shown in Fig. 5.14. In the cases considering the robot collisions, the performance of the robotic swarm decreased with increasing the robot size, as shown in Fig. 5.14. These results are because the robots are more likely to interfere with each other in a larger robot size, and fewer robots can enter the target area at the same time. In contrast, the performance of the robotic swarm without considering the robot collisions is kept at high values regardless of the robot size (see

[*6] The experiments without collisions are implemented using *collision filtering* in Box2D. The collision filtering will allow the robots to collide with walls but prevent colliding with each other. For further details on the collision filtering, see the "Filtering" section in the "Dynamics Module" page of the Box2D documentation (`https://box2d.org/documentation/md__d_1__git_hub_box2d_docs_dynamics.html`).

also Fig. 5.14). When comparing the performance with and without considering the robot collisions, the robotic swarms without the robot collisions scored the higher fitness values.

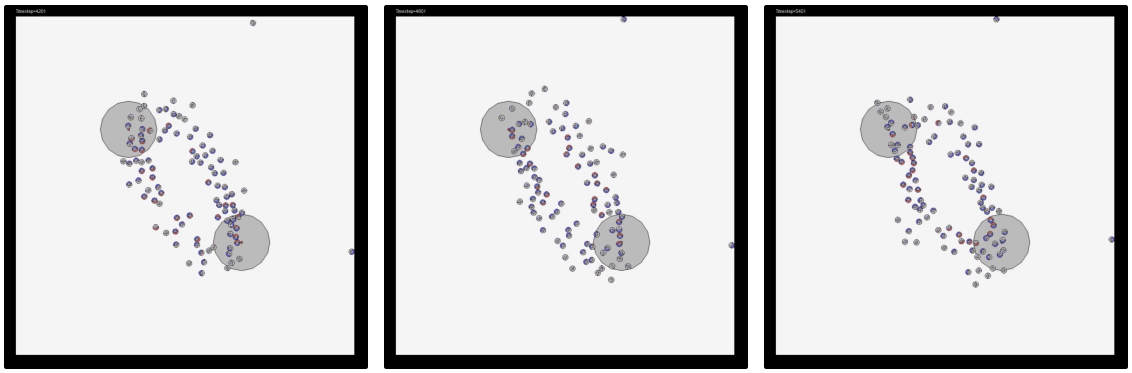
The snapshots of the behavior observed using the robotic swarms that consider robot collisions are shown in Figs. 5.15, 5.16, and 5.17. The robotic swarm with 0.1 and 0.2 m diameter robots exhibit a specialization among robots; i.e., the robots traveling the inside of the path activate their LEDs, while those traveling outside deactivate them (see also Figs. 5.15 and 5.16). From the observed behavior, the rear LEDs seem to be used as a guide to form the path, while the front LEDs are used to avoid collisions with robots traveling in the opposite direction. However, in the behavior using the 0.4 m diameter robots, the robots tend to activate the rear LEDs when traveling the outer side of the path (see also Fig. 5.17). Furthermore, some robots decide to stay near the wall and not to join the swarm. These robots seem to be providing space to the other robots to mitigate congestion and improve the performance of the robotic swarm as a whole.

The snapshots of the behavior using the robotic swarms without considering the robot collisions are shown in Figs. 5.18, 5.19, and 5.20. Compared to the robotic swarms with the robot collisions, the behavior observed without robot collisions exhibits a coherent path, as can be seen from Figs. 5.18, 5.19, and 5.20. Moreover, the robotic swarms showed similar behavior to form a path without performing behavioral specialization, regardless of the robot size.

5.3.2 Discussion

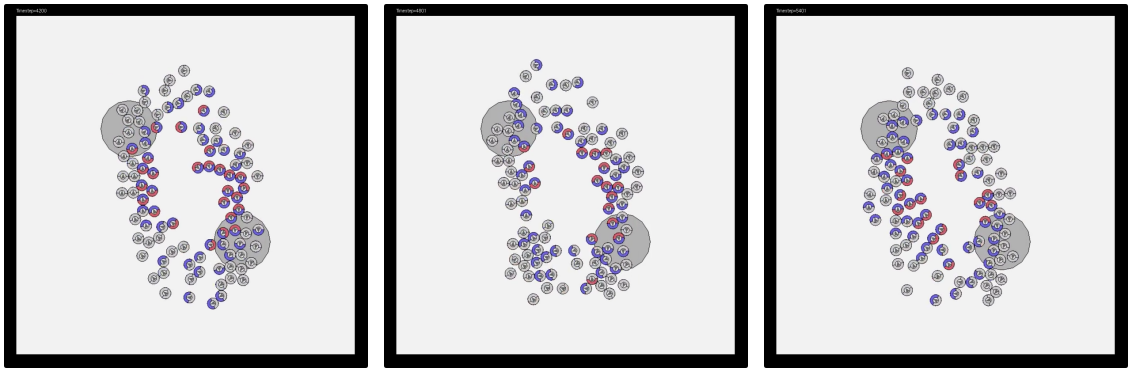
The activation rates of the LEDs are calculated to discuss the behavior of the robotic swarm. For each robot, the activation rates of the LEDs during the 1200 to 7200 time steps are calculated by Eq. (4.5). The scatter plots of the front versus rear LED activation rates are shown in Figs. 5.21 and 5.22. In addition to the activation rates, the colors in Figs. 5.21 and 5.22 show the sub-fitness f_n in Eq. (4.3), which indicates the number of times the corresponding robot entered the landmarks during 1200 to 7200 time steps.

The scatter plots of the LED activation rates using robots with considering collisions are shown in Fig. 5.21. Along with the scatter plots with 0.2 and 0.4 m diameter robots, four robots with different activation rates are selected, and their trajectories within the environment are plotted in Fig. 5.23. In the robotic swarm with 0.1 and 0.2 m diameter robots, the distributions of the activation rates show a positive correlation between the two activation rates, as shown in Figs. 5.21(a) and (b). Moreover, robots with lower activation rates tend to have lower sub-fitness f_n. The robots traveling outside of the path are inefficient in performing the task, and therefore, they tend to obtain lower sub-fitness f_n. The robotic swarm with 0.1 and 0.2 m diameter robots exhibits specialization; i.e., the robots traveling inside activate their LEDs while those traveling the outside deactivate them (see



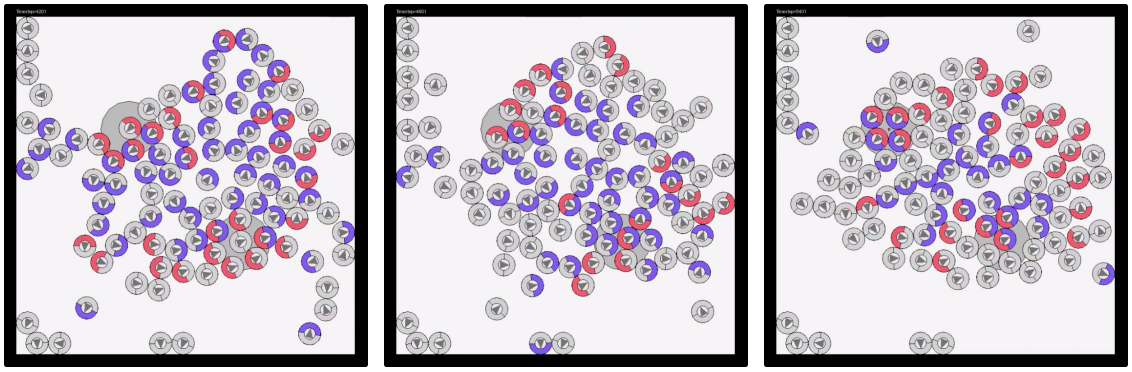
(a) 4200 time steps. **(b)** 4800 time steps. **(c)** 5400 time steps.

Fig. 5.15. Snapshots of behavior observed using the robots with the diameter of 0.1 m and with robot collisions.



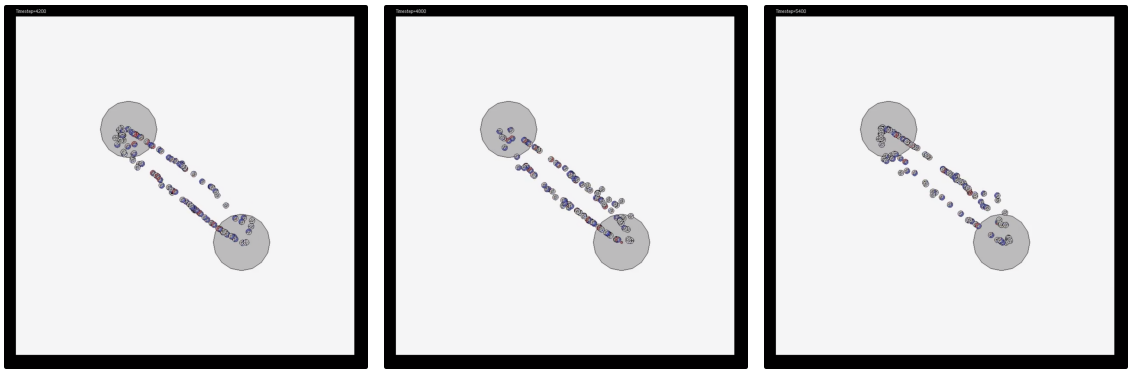
(a) 4200 time steps. **(b)** 4800 time steps. **(c)** 5400 time steps.

Fig. 5.16. Snapshots of behavior observed using the robots with the diameter of 0.2 m and with robot collisions.



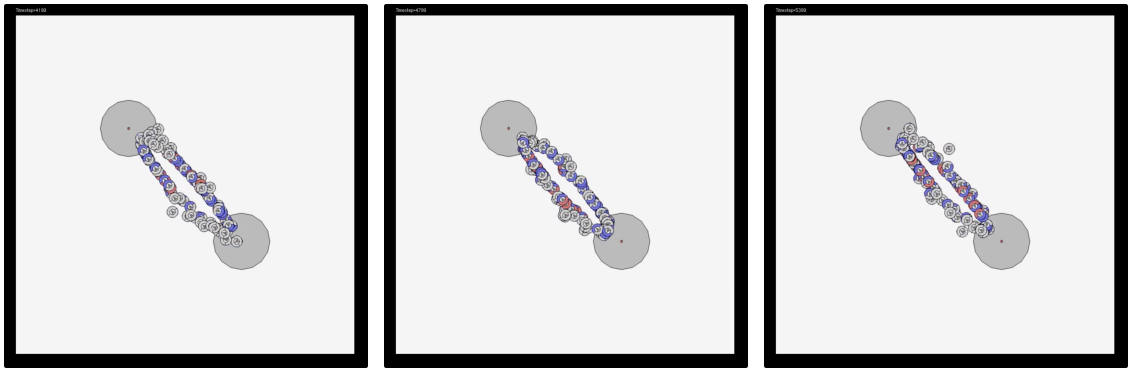
(a) 4200 time steps. **(b)** 4800 time steps. **(c)** 5400 time steps.

Fig. 5.17. Snapshots of behavior observed using the robots with the diameter of 0.4 m and with robot collisions.



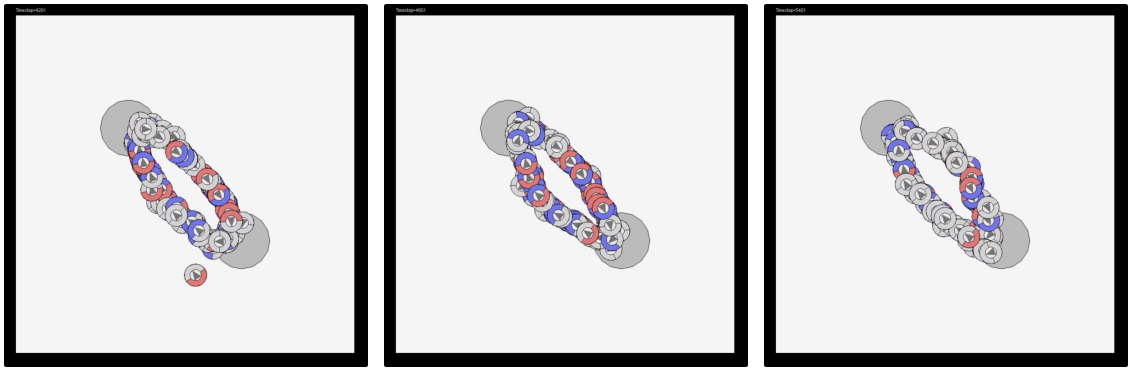
(a) 4200 time steps. **(b)** 4800 time steps. **(c)** 5400 time steps.

Fig. 5.18. Snapshots of behavior observed using the robots with a diameter of 0.1 m and without robot collisions.



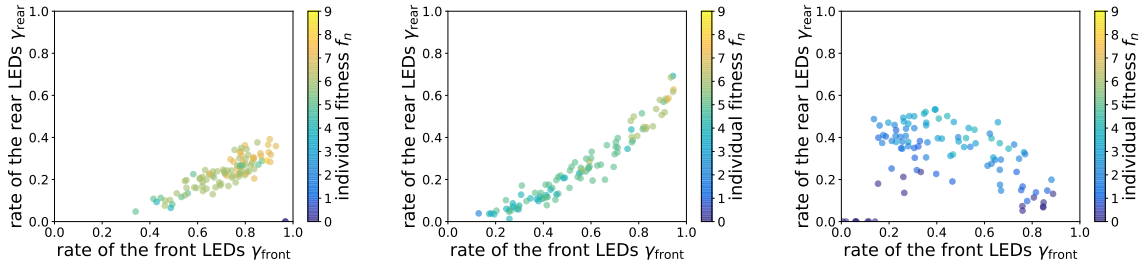
(a) 4200 time steps. **(b)** 4800 time steps. **(c)** 5400 time steps.

Fig. 5.19. Snapshots of behavior observed using the robots with a diameter of 0.2 m and without robot collisions.



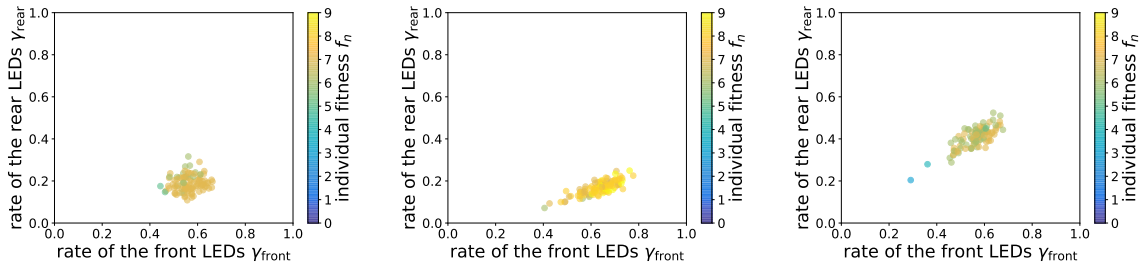
(a) 4200 time steps. **(b)** 4800 time steps. **(c)** 5400 time steps.

Fig. 5.20. Snapshots of behavior observed using the robots with a diameter of 0.4 m and without robot collisions.



(a) 0.1 m. **(b)** 0.2 m. **(c)** 0.4 m.

Fig. 5.21. Scatter plots of the activation rate of the LEDs during 1200–7200 time steps with robot collisions. Each point indicates a robot with corresponding activation rates. The color of the point shows the sub-fitness value f_n of the corresponding robot.



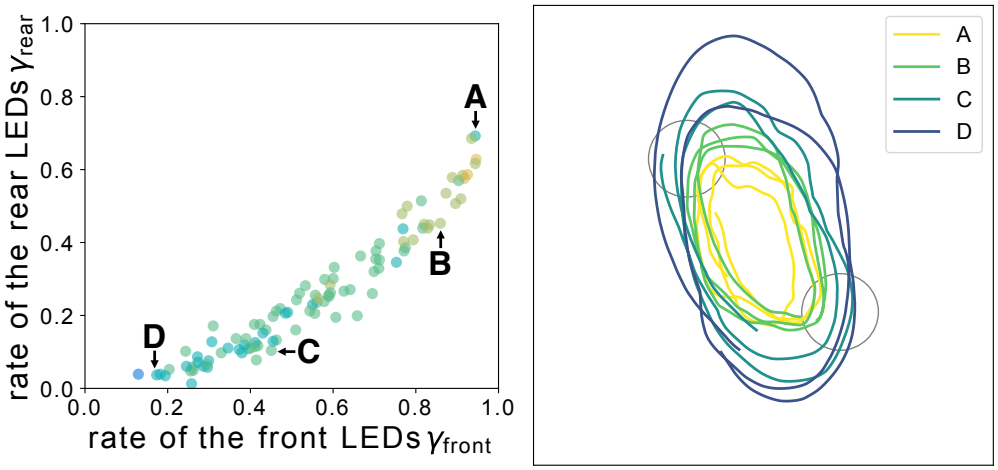
(a) 0.1 m. **(b)** 0.2 m. **(c)** 0.4 m.

Fig. 5.22. Scatter plots of the activation rate of the LEDs during 1200–7200 time steps without robot collisions.

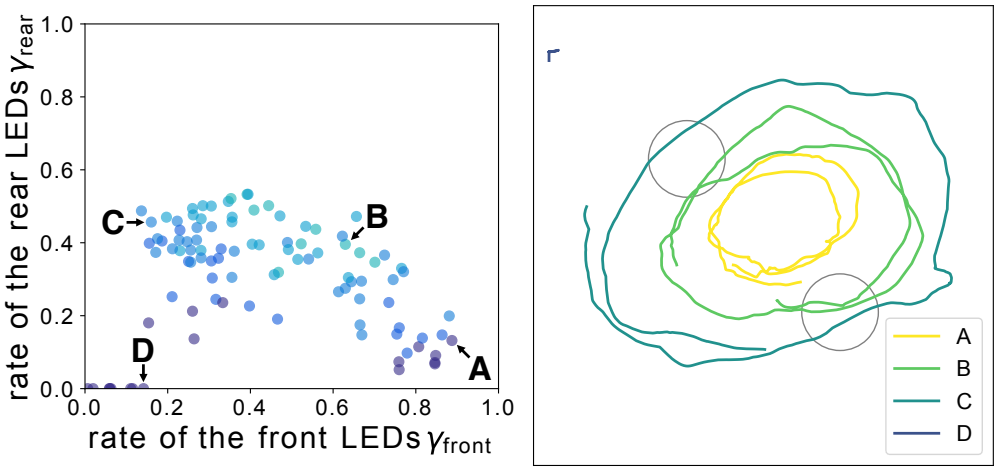
also Fig. 5.23(a)).

In the case of the robotic swarm with 0.4 m diameter robots, the higher sub-fitness values are obtained by the robots with moderate activation rates (see also Fig. 5.21(c)). The robots with a higher γ_{front} and a lower γ_{rear} traveled the inner side of the path, as can be seen from Fig. 5.23(b). However, these robots traveled too far inward and failed to enter the landmarks. The robots with a lower γ_{front} and a higher γ_{rear} traveled the outer side of the path, which is more inefficient in performing the task. The robots with low γ_{front} and γ_{rear} correspond to those that stay near the wall deactivating their LEDs. Therefore, these results show that behavioral specialization has emerged in robotic swarms in situations considering robot collisions.

The results using robots without considering collisions are shown in Fig. 5.22. Compared to the scatter plots of the experiments considering the robot collisions, the results without collisions show a coherent distribution (see also Figs. 5.21 and 5.22). The robotic swarms with 0.2 and 0.4 m diameter robots show a slight positive correlation, as shown in Figs. 5.22(b)



(a) 0.2 m diameter with robot collisions.



(b) 0.4 m diameter with robot collisions.

Fig. 5.23. Trajectories of the selected robots with corresponding LED activation rates from the 1200 to 7200 time steps.

and (c). This positive correlation is due to the timing of each robot joining the path, i.e., the robots that were late in joining the path had slightly lower activation rates. Furthermore, almost all robots obtained relatively similar sub-fitness values f_n in the cases without considering collisions. These results imply that all robots have a similar strategy for activating LEDs; i.e., the specialization seems not to emerge in a swarm without robot collisions.

It can be assumed that collisions among robots provide feedback on a robotic swarm to emerge specialization. For example, ants exhibit priority rules to avoid congestion among

individuals in crowded conditions [27, 38]. Similar to the priority rules in ants, the robotic swarm with robot collisions exhibits the specialization to manage congestion. In particular, the collisions among robots lead to constraints on the mobility of robots, and therefore, the emergent specialization makes it possible to perform the task more efficiently in congested situations. In contrast, the robotic swarm without collisions does not show behavioral specialization because there is no need to deal with congestion. The robot collisions are sometimes neglected in studies of swarm robotics, especially in experiments conducted using computer simulations. Therefore, it can be concluded that the robot collisions are also an essential feature to discuss the performance of robotic swarms.

5.4 Conclusions

This chapter discussed the effects of congestion on swarm performance by considering the number of robots and the robot size. The controllers of the robotic swarm were designed using the evolutionary robotics approach. The results showed that also the size of robots was an essential feature to discuss the relationship between swarm performance and congestion. Also, this chapter discussed the effects of collisions among robots on the collective behavior in robotic swarms. The experiments were conducted in computer simulations with and without considering robot collisions. The results showed that the robot collisions would affect the performance of the robotic swarm. In addition, the results showed that the collisions among robots provided feedback on robotic swarms to exhibit behavioral specialization, which did not emerge in situations without robot collisions.

Chapter 6

Evolving Echo State Networks for Generating Collective Behavior of a Robotic Swarm

This chapter proposes an evolutionary robotics approach for designing the controller that is based on the echo state network [58, 80]. Echo state networks are an alternative to the traditional recurrent neural networks, which are suitable for practical applications that use time-series data. A hidden layer of an echo state network consists of a recurrent neural network with sparsely and randomly generated connections. One of the main characteristics of the echo state network is in the training process, i.e., it only tunes the output weight values, which leads to speeding up the training. This chapter applies an evolutionary algorithm to optimize the output weight values of the echo state network. The echo state network is employed as the controller of the robot. The performance of the robotic swarm is evaluated in a path-formation task [54, 104], which aims to develop a path of robots between two landmarks and navigate between them. The performance of the controller using the echo state network is compared with the traditional neural network controller.

The chapter is organized as follows. Section 6.1 briefly introduces the echo state network. Section 6.2 describes the task to be performed by the robotic swarm. Section 6.3 presents the settings of the evolutionary robotics approach. Section 6.4 shows the results obtained in the experiments. Finally, Section 6.5 concludes this chapter.

6.1 Echo State Networks

The echo state network is one of the pioneering methods in a computational framework called *reservoir computing* [58, 80]. The basic idea of the echo state network is shared with the liquid state machine [81], which is the other pioneering method in reservoir computing.

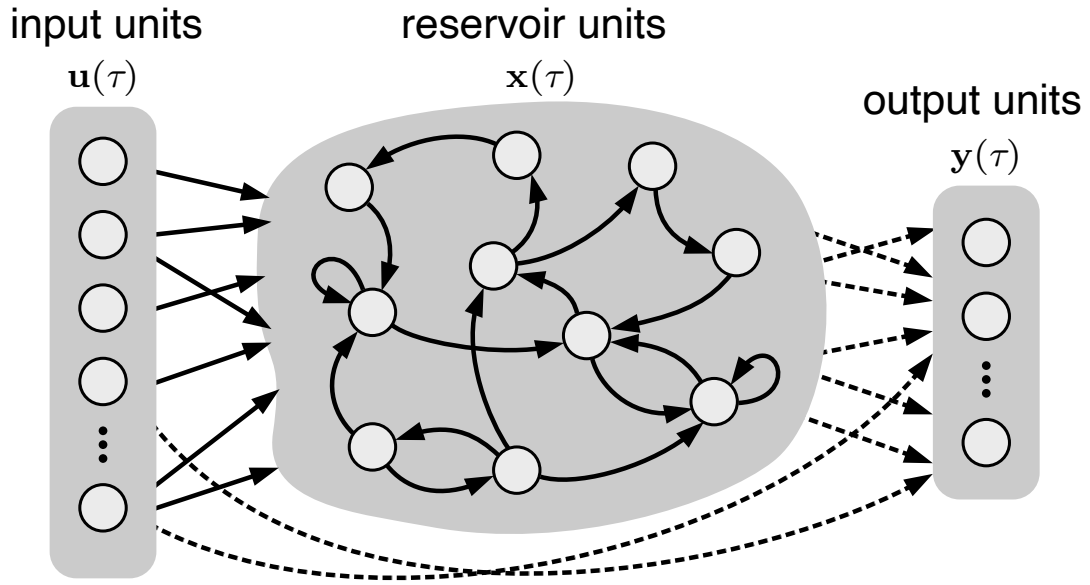


Fig. 6.1. The basic structure of an echo state network. Solid arrows indicate fixed connections. Dashed arrows are trainable readout connections.

In reservoir computing, a recurrent neural network that is generated with a random topology is used as a so-called *reservoir*. The connections in the reservoir are fixed, and only the readout from the reservoir is trained.

The basic structure of an echo state network is illustrated in Fig. 6.1. Let an echo state network have N_{res} reservoir units, N_{in} input units, and N_{out} output units. The states of the reservoir are updated by the following equation:

$$\mathbf{x}(\tau) = \varphi_{\text{res}}\left(\mathbf{W}_{\text{in}}\mathbf{u}(\tau) + \mathbf{W}_{\text{res}}\mathbf{x}(\tau - 1)\right), \tag{6.1}$$

where $\mathbf{x}(\tau) \in \mathbb{R}^{N_{\text{res}}}$ is the vector of the reservoir units at time step τ, $\mathbf{u}(\tau) \in \mathbb{R}^{N_{\text{in}}}$ is the vector of the input signals at time step τ, $\varphi_{\text{res}}(\cdot)$ is the activation function of the reservoir units, $\mathbf{W}_{\text{in}} \in \mathbb{R}^{N_{\text{res}} \times N_{\text{in}}}$ is the input weight matrix, and $\mathbf{W}_{\text{res}} \in \mathbb{R}^{N_{\text{res}} \times N_{\text{res}}}$ is the reservoir weight matrix. Typically, a sigmoid or hyperbolic tangent function is used as the activation function $\varphi_{\text{res}}(\cdot)$. The input weight matrix $\mathbf{W}_{\text{in}}$ is usually set to have dense connections that are generated randomly from a uniform distribution. In contrast, the reservoir weight matrix $\mathbf{W}_{\text{res}}$ is generated with sparse and random connections. The output of the echo state network is obtained by the following equation:

$$\mathbf{y}(\tau) = \varphi_{\text{out}}\left(\mathbf{W}_{\text{out}}\left[\mathbf{u}(\tau)\,;\mathbf{x}(\tau)\right]\right), \tag{6.2}$$

where $\mathbf{y}(\tau) \in \mathbb{R}^{N_{\text{out}}}$ is the vector of the output units at time step τ, $\varphi_{\text{out}}(\cdot)$ is the activation function of the output units, $\mathbf{W}_{\text{out}} \in \mathbb{R}^{N_{\text{out}} \times (N_{\text{in}} + N_{\text{res}})}$ is the output weight matrix, and $[\mathbf{u}(\tau)\,;\mathbf{x}(\tau)]$ indicates the vertical concatenation of $\mathbf{u}(\tau)$ and $\mathbf{x}(\tau)$. The identity or a sigmoid function is commonly used as the output activation function $\varphi_{\text{out}}(\cdot)$. Only the weight matrix

$\mathbf{W}_{\text{out}}$ is trained in the echo state network. Typically in supervised learning, $\mathbf{W}_{\text{out}}$ is trained to minimize the error between the output from the network and the desired output.

In the traditional echo state network, the reservoir is generated to have a large number of units, with sparse and random connections. In general, the number of reservoir units N_{res} is generated with the order of tens to thousands. The weight matrix $\mathbf{W}_{\text{res}}$ is generated with several to 20 percent of possible connections and with weight values generated randomly from a uniform distribution symmetric around zero. The essential point of the echo state network is that the reservoir should have the *echo state property* [57]. This property implies that the effect of previous states of the reservoir and input units will be asymptotically eliminated as time passes without persisting or being amplified. In most situations, the echo state property is obtained if the reservoir weight matrix $\mathbf{W}_{\text{res}}$ is scaled to have $\rho(\mathbf{W}_{\text{res}}) < 1$, where $\rho(\cdot)$ is the spectral radius (i.e., the largest absolute value of the eigenvalues) of a matrix. The spectral radius of the reservoir $\rho(\mathbf{W}_{\text{res}})$ should be close to 1 if the task requires a long memory of the inputs.

6.2 Settings of the Path-formation Task

A path-formation task is one of the fundamental tasks addressed in the study of swarm robotics [4, 10, 104]. In this task, the goal of a robotic swarm is to develop a collective path of robots and navigate between two landmarks. The details of the path-formation task and the settings of the robot are described in Section 4.1. In this chapter, 25 robots are employed to address the path-formation task. The experiments are carried out in computer simulations using the *Box2D* physics engine [15].

6.3 Evolutionary Robotics Approach

In this chapter, the *covariance matrix adaptation evolution strategy* [49, 51] is employed to optimize the controller. The covariance matrix adaptation evolution strategy provides recommendation values for the hyperparameters; therefore, only the population size and the initial mutation step size are set depending on the objective. The population size and the initial mutation step size are set to $\lambda = 300$ and $\sigma = 0.2$, respectively. The evolutionary process lasts for 1000 generations, with the zeroth generation of a randomly generated population.

The evaluation of a controller is done based on the predefined fitness function, which indicates the achievement of the task addressed by the robotic swarm. The fitness function for the path-formation task is described in Section 4.2.3.

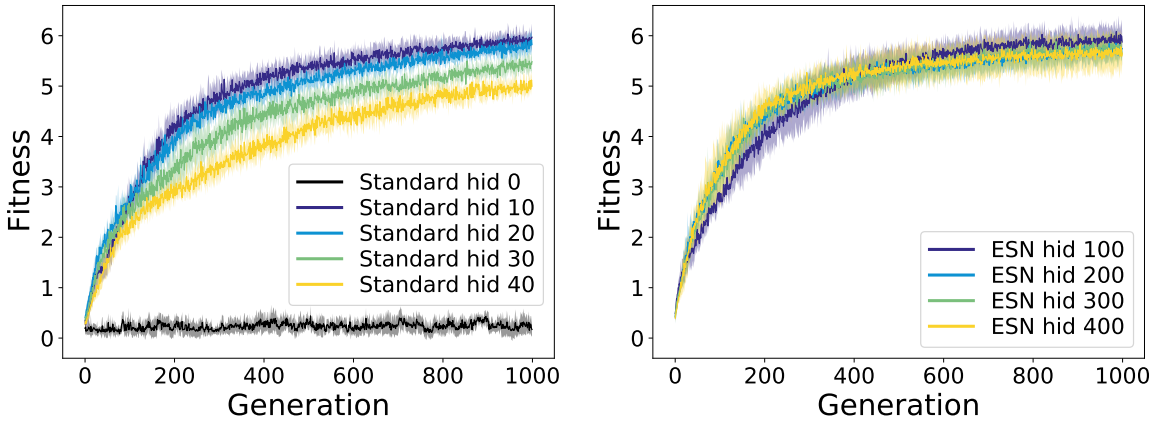


Fig. 6.2. Transitions of the highest fitness values obtained for the five evolutionary processes. Solid lines indicate the mean of the highest fitness values over the five evolutionary processes, while shaded areas correspond to the standard deviations. The settings with "Standard" (left) represent the results using the traditional evolutionary robotics approach and "ESN" (right) for the proposed method using the echo state network. The number after the "hid" indicates the number of units in the hidden layer or in the reservoir.

6.4 Experiments and Results

In this chapter, an echo state network is implemented as a controller for the robotic swarm. The covariance matrix adaptation evolution strategy only optimizes the output weight matrix $\mathbf{W}_{\text{out}}$. The input weight matrix $\mathbf{W}_{\text{in}}$ is generated with full connections and with the weight values that are randomly chosen from $\{-1, 1\}$. The reservoir weight matrix $\mathbf{W}_{\text{res}}$ is randomly generated with 10 percent of possible connections. The weight values of $\mathbf{W}_{\text{res}}$ are also chosen from random values in $\{-1, 1\}$, and after that, they are scaled to have the spectral radius $\rho(\mathbf{W}_{\text{res}}) = 0.9$. The hyperbolic tangent and sigmoid functions are used for the activation functions $\varphi_{\text{res}}(\cdot)$ and $\varphi_{\text{out}}(\cdot)$, respectively. The number of reservoir units N_{res} is varied by 100, 200, 300, and 400.

For comparison with the echo state network, the experiments are conducted using the traditional evolutionary robotics approach. The controller for the robot is represented by a fully connected recurrent neural network with evolving all of the weight values. The number of hidden units for the recurrent neural network is set to 10, 20, 30, and 40. Additionally, the experiments are conducted using a fully connected feed-forward neural network that only has input to output connections.

6.4.1 Results

Five independent evolutionary processes are executed for each neural network controller with a different random seed. The transitions of the highest fitness values are shown in Fig. 6.2.

Table 6.1. Number of weight values optimized by the evolutionary algorithm.

Settings	Number of weights
Standard hid 0	80
Standard hid 10	420
Standard hid 20	960
Standard hid 30	1700
Standard hid 40	2640
ESN hid 100	480
ESN hid 200	880
ESN hid 300	1280
ESN hid 400	1680

In addition, the total number of weight values optimized by the evolutionary algorithm is listed in Table 6.1. The fitness values of the controller with the two-layered feed-forward neural network ("Standard hid 0" in Fig. 6.2) seem to stagnate around zero throughout the evolutionary process. Since the feed-forward neural network has no mechanism to store information about previous inputs, this result implies that the controller has to handle the time-series data to accomplish the path-formation task. The controller using the echo state networks (ESN) and the recurrent neural network with 10 and 20 hidden units (Standard hid 10 and hid 20) have a similar fitness progression with the highest fitness values converging toward 6. On the other hand, the controller using the recurrent neural network with 30 and 40 hidden units (Standard hid 30 and hid 40) showed a slower convergence of fitness values. These results are because more units in the hidden layer will increase the number of parameters to be optimized by an evolutionary algorithm, and therefore, they show a slower convergence of the fitness values. However, when comparing the echo state network using 400 reservoir units (ESN hid 400) with the recurrent neural network using 30 hidden units (Standard hid 30), despite having a similar quantity of weights to be optimized, the echo state network showed a faster convergence of the fitness value.

For each controller setting, the set of weight values that have obtained the highest fitness value is selected and re-evaluated for 100 trials. The results of the re-evaluation are shown in Fig. 6.3. In the results using the traditional evolutionary robotics approach ("Standard" in Fig. 6.3), the controller with 10 hidden units showed the highest performance, and the performance decreased with increasing the number of hidden units. When comparing the controller composed of 10 hidden units (Standard hid 10) with the echo state networks (ESN), there is no significant difference between the results obtained using the controller with 200 or 300 reservoir units (Mann-Whitney U test, p-value > 0.05). However, the recurrent neural

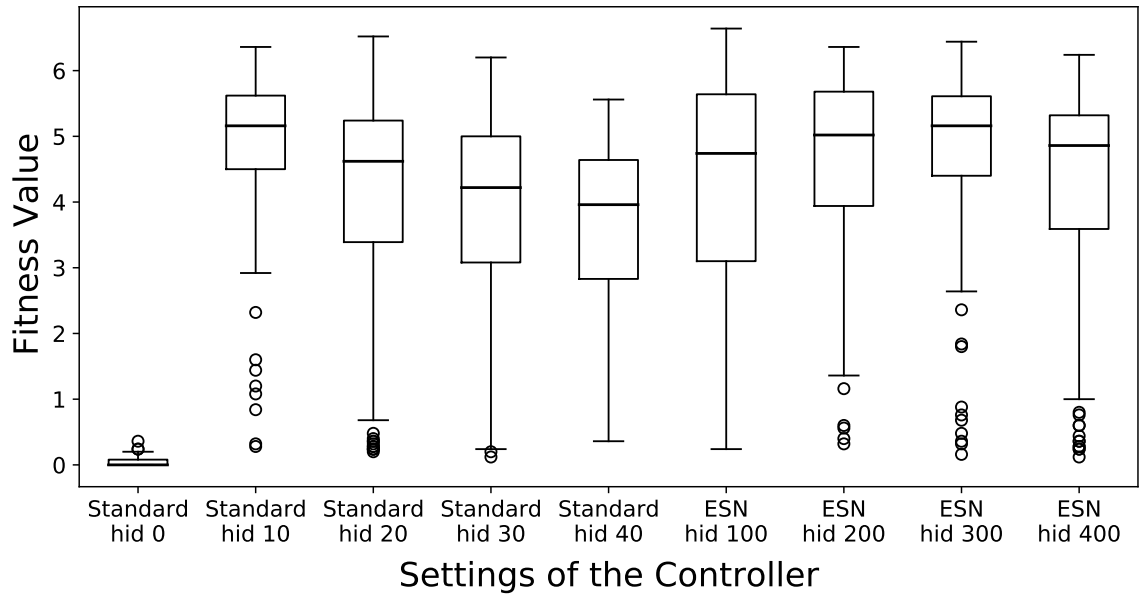


Fig. 6.3. Results of the re-evaluation of the best-evolved controller with different settings over 100 trials.

network controller with 10 hidden units obtained a significantly higher performance than the controller using the echo state network with 100 or 400 reservoir units (p-value < 0.05). As for the results using the controllers with the echo state network (ESN), the performance is kept at relatively high values regardless of the number of reservoir units.

The overall results indicate that the echo state network could be an alternative method for the controller of the robotic swarm in the evolutionary robotics approach. The controller using the echo state network with 200 or 300 reservoir units showed similar performance to the recurrent neural network with 10 hidden units. The advantage of the echo state network could be the simplicity of the parameter settings; that is, the number of reservoir units seems to have a lower influence on performance than that of the recurrent neural networks. Furthermore, the echo state networks seem to have a faster convergence in optimizing controllers than traditional recurrent neural networks.

6.5 Conclusions

This chapter proposed the echo state network as an alternative to the recurrent neural network in the evolutionary robotics approach for designing a controller for a robotic swarm. The controllers for a robotic swarm were optimized by the covariance matrix adaptation evolution strategy. The performance of the controller was discussed using the path-formation task addressed by the robotic swarm. The results confirmed that the controller using the echo state network performed as well as the traditional recurrent neural network. Moreover, the

results showed that the echo state network could reduce the effort required in designing the parameters of the robot controller. Besides, the echo state networks had a faster convergence in optimizing controllers than the traditional recurrent neural networks.

Chapter 7

Topology and Weight Evolving Artificial Neural Networks in Cooperative Transport by a Robotic Swarm

This chapter focuses on designing controllers for a robotic swarm using Topology and Weight Evolving Artificial Neural Networks (TWEANNs). The TWEANN algorithm optimizes both the synaptic weight values and the topological structure of the neural network. Since the traditional evolutionary robotics approach uses a neural network with fixed topology, it might restrict the robot's behavior or have unsuitable structures within the controller. TWEANNs could be an alternative to overcome these limitations in the traditional evolutionary robotics approach. However, so far, there are only a few studies that have applied TWEANN to designing controllers for a robotic swarm (e.g., [26, 44]).

This chapter uses a TWEANN algorithm that only applies mutations to evolve neural networks, namely Mutation-Based Evolving Artificial Neural Network (MBEANN) [91], to design the controller for a robotic swarm. In addition, NeuroEvolution of Augmenting Topologies (NEAT) [106], which is a widely used TWEANN algorithm, is employed for comparison. The controllers are evaluated in a cooperative transportation task, in which robots have to cooperate to transport heavy objects to the goal. The results show that the MBEANN approach could be applied to designing a controller for a robotic swarm. Moreover, the robot controller designed by the MBEANN approach outperforms NEAT in the cooperative transportation task.

This chapter is organized as follows. Section 7.1 presents an overview of MBEANN. Section 7.2 describes the settings of the cooperative transportation task that is performed

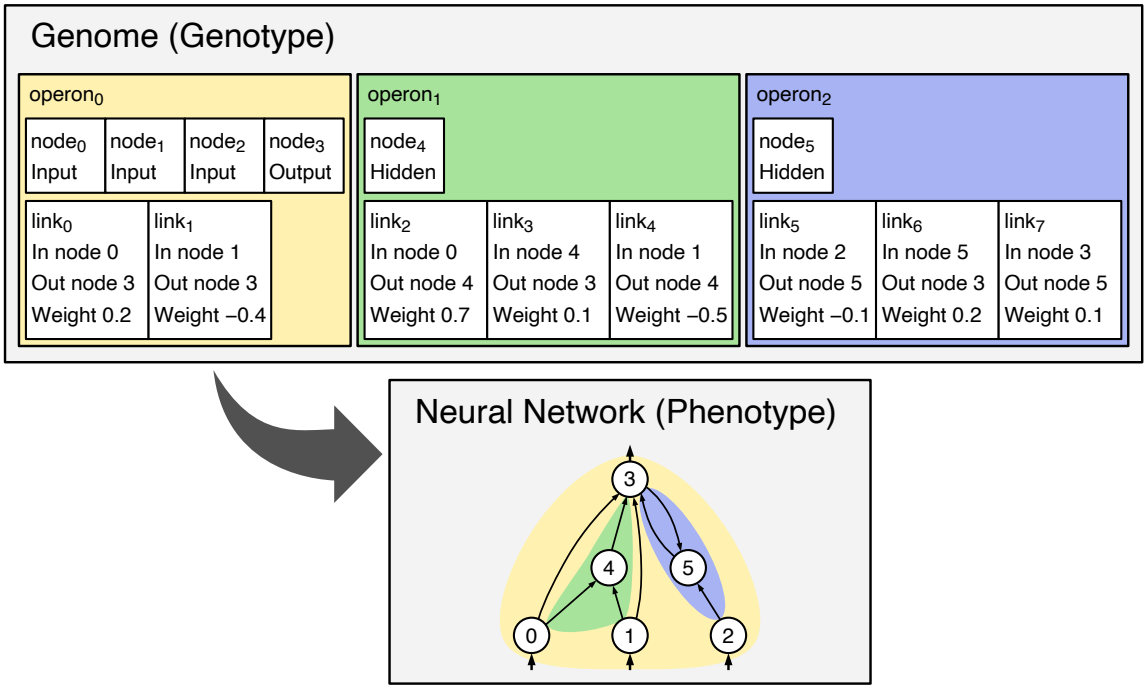


Fig. 7.1. Example of a genotype to phenotype mapping in MBEANN.

by a robotic swarm. Section 7.3 shows the results obtained in the experiments, along with a discussion. Finally, Section 7.4 concludes this chapter.

7.1 Mutation-Based Evolving Artificial Neural Network (MBEANN)

The MBEANN algorithm is proposed as an alternative method of TWEANN [91]. This algorithm only uses mutations and no crossover between individuals. Structural mutations in MBEANN are designed to have nearly or completely neutral. In other words, the signal transfer of the input to output mapping of the neural network will only have small or no changes before and after the structural mutations. Additionally, an individual of MBEANN is designed to have a set of subnetworks. That is, the topology of the neural network grows independently within each subnetwork. The MBEANN algorithm developed and used in this study is open source and available at [53]. The rest of this section describes the encoding method and mutation operators of MBEANN.

7.1.1 Encoding Method

An individual in MBEANN is designed to have a set of subnetworks, which are called *operons*. An example of a basic structure of an individual for MBEANN is illustrated in Fig. 7.1. As shown in Fig. 7.1, each genome is composed of a set of operons. Moreover,

each operon is composed of a set of nodes and connections between the nodes. Each node is assigned with the identification number and the node type. The node type indicates to which layer the node belongs. A connection is also called a link, which consists of the identification number, the identification number of the input node, the identification number of the output node, and the weight value. Hence, a genotype in MBEANN could be defined as follows:

$$\text{genome} = \{\text{operon}_0, \text{operon}_1, \ldots, \text{operon}_M\}, \tag{7.1}$$

$$\text{operon}_i = \{\{\text{node}_j | \text{node}_j \in O_{Ni}\}, \{\text{link}_k | \text{link}_k \in O_{Li}\}\}, \tag{7.2}$$

where O_{Ni} is the set of nodes in operon_i, O_{Li} is the set of links in operon_i, and M is the number of operons. The operon_0 only consists of the input nodes, the output nodes, and the connections between them. The individuals in the population are initialized with only having the operon_0, which is the minimal structure without hidden nodes. As for $\text{operon}_i (i \neq 0)$, it consists of one or more hidden nodes with a set of links. The links in operon_i connect between nodes within the operon_i or a node in the operon_i and a node in the operon_0.

7.1.2 Mutation Operators

The MBEANN algorithm employs two structural mutations, i.e., the add-node mutation and the add-connection mutation. Additionally, synaptic weights are mutated by applying a small change to each weight value. The mutation operators of MBEANN are defined as follows.

Add-node Mutation

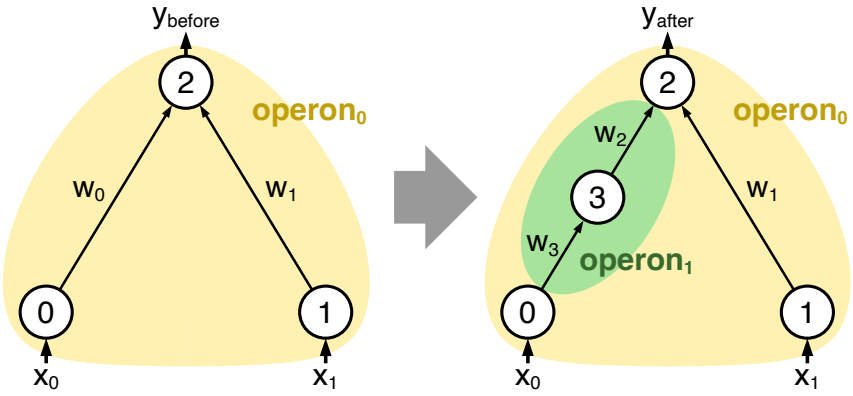
The add-node mutation is applied to each operon with a probability of p_{node}. This mutation selects one connection from the operon and replaces it with a new node and two new connections. If the mutation is applied to operon_0, the new node and connections are assigned to a new operon. The pseudo-code of the add-node mutation is described in Algorithm 7.1.

The structural mutations of MBEANN are designed to keep the signal transfer of the input to output mapping of the neural network. An example of the add-node mutation is illustrated in Fig. 7.2(a). Suppose that the activation function $\varphi(\cdot)$ is applied to each hidden and output node. Let the link_i has the weight value of w_i. In the case of Fig. 7.2(a), the output before and after the mutation could be described as follows:

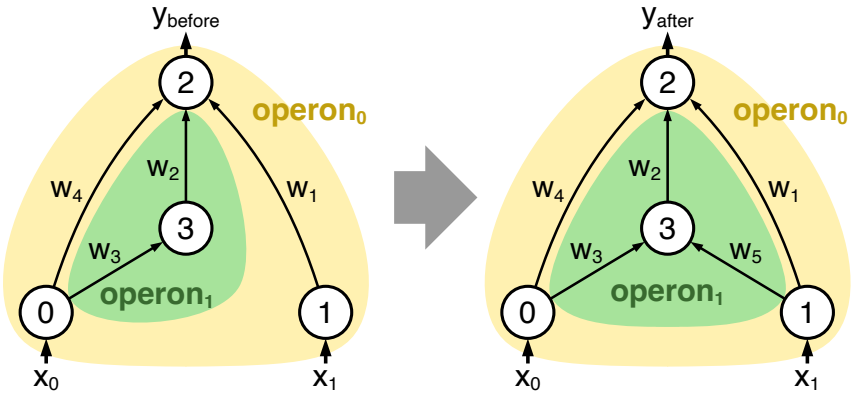
$$y_{\text{before}} = \varphi(w_0 x_0 + w_1 x_1), \tag{7.3}$$

$$y_{\text{after}} = \varphi(w_2 \varphi(w_3 x_0) + w_1 x_1). \tag{7.4}$$

The add-node mutation should be designed to satisfy $y_{\text{before}} \simeq y_{\text{after}}$. Thus, if assuming that $w_0 = w_2$, then $y_{\text{before}} \simeq y_{\text{after}}$ is described as $x_0 \simeq \varphi(w_3 x_0)$. Moreover, let $\mathcal{L}(\cdot)$ be the error between before and after the mutation, and if the activation function is defined as



(a) Add-node mutation.



(b) Add-connection mutation.

Fig. 7.2. Example of structural mutations in MBEANN.

$\varphi(x) = 1/\left(1 + e^{\beta(\alpha - x)}\right)$, the error is described as follows:

$$\mathcal{L}(x_0) = x_0 - \varphi(w_3 x_0) = x_0 - \frac{1}{1 + e^{\beta(\alpha - w_3 x_0)}}, \tag{7.5}$$

where α is the midpoint of the sigmoid activation function $\varphi(x)$ and β determines the steepness of $\varphi(x)$. Taking into account that $0 < \varphi(x) < 1$ and assuming that the input values of the neural network are normalized within the range $[0, 1]$, the parameters of $\varphi(x)$ are set to have $\alpha = 0.50 w_3$ and $\beta = 4.63/w_3$ $(w_3 \neq 0)$. For simplicity, w_3 is set to 1.0.

Add-connection Mutation

In the add-connection mutation, a new connection is created to connect two previously unconnected nodes. The pseudo-code of the add-node connection is described in Algorithm 7.2, along with an example illustrated in Fig. 7.2(b). The add-connection mutation is applied to each operon with a probability of p_{link}. If the mutation is applied to the operon$_i$, a new connection is created between two nodes in the operon$_i$ or between one of each

Algorithm 7.1: Pseudo-code of the add-node mutation.

```
foreach operon in genome do
    if rand(0,1) < p_node then
        // rand(0,1) is a uniform random number between 0 and 1
        randomly select one connection from the operon;
        if selected connection is in operon_0 then
            remove the selected connection;
            add a new node and two new connections to the new operon;
        else
            remove the selected connection;
            add a new node and two new connections to the current operon;
        end
    end
end
```

Algorithm 7.2: Pseudo-code of the add-connection mutation.

```
foreach operon in genome do
    if rand(0,1) < p_link then
        randomly select one node from the operon;
        randomly select another node from the current operon or from operon_0;
        add a new link with the weight value w = 0;
    end
end
```

from the operon_i and the operon_0. In other words, connections between operon_i and operon_j ($i \neq j, i \neq 0, j \neq 0$) are prohibited. Therefore, operons will grow independently from each other, which promotes the neural network to have a modular structure. The weight value of the newly generated connection is set to zero to ensure the neutrality of the mutation.

For the implementation in a computer program [53], each operon has a list of unestablished connections. When the mutation is applied to the operon, randomly select one element from the list and establish it as the new connection. Providing the list of unestablished connections will make it easy to handle the add-link mutation, especially for the neural network with directed connections.

Synaptic Weight Mutation

Each weight value is perturbed by adding a random value with a probability of p_{weight}. The Gaussian distribution with a mean of zero and a standard deviation of σ is employed for generating the random value. In this chapter, the standard deviation is set to $\sigma = 0.05$.

7.2 Cooperative Transport by a Robotic Swarm

This section describes the settings of the cooperative transportation task addressed by a robotic swarm. Cooperative transport is one of the fundamental tasks that is often studied in the field of swarm robotics [4, 10, 116]. In this task, robots have to coordinate their actions to transport objects that are too heavy to be moved by a single robot. The experiments are conducted using *pybox2d* [72], a Python library for simulating 2D physics based on the Box2D physics engine.

7.2.1 Task Environment

The environment of the cooperative transportation task is illustrated in Fig. 7.3. The arena is surrounded by walls. Twenty robots and two objects are placed in the arena. The initial positions of robots are randomly determined and placed facing a fixed direction. The two objects are initially located at fixed positions. The mass of the object is set to 20 mass units.[*7] Each object requires at least five robots to start moving, with all five robots pushing it in the same direction. When the center of an object has reached the goal line, a new object will be generated at the initial position. The robotic swarm should transport the objects toward the goal line as many as possible within the time limit.

7.2.2 Robot Settings

The sensor settings of the robot are illustrated in Fig. 7.4. The robot has a circular body and moves with a two-wheeled differential drive method. The robot has an omnidirectional camera, seven distance sensors, and an electronic compass.

The sensor range of the omnidirectional camera is divided into four slices, as can be seen in Fig. 7.4. The omnidirectional camera only detects the existence of robots and objects within each slice. The process for getting sensor values from the omnidirectional camera is done independently for detecting robots and objects. Each slice in the omnidirectional camera returns 1 if a robot or object has been detected and 0 otherwise.

Seven distance sensors are attached to the front side of the robot. The distance sensor can detect walls, objects, and other robots within the sensor range. The distance sensor returns

[*7] The Box2D is tuned to work well with meter-kilogram-second (MKS) units with simulating 60 time steps per second. Hence, the object has a mass of 20 kg in the simulation.

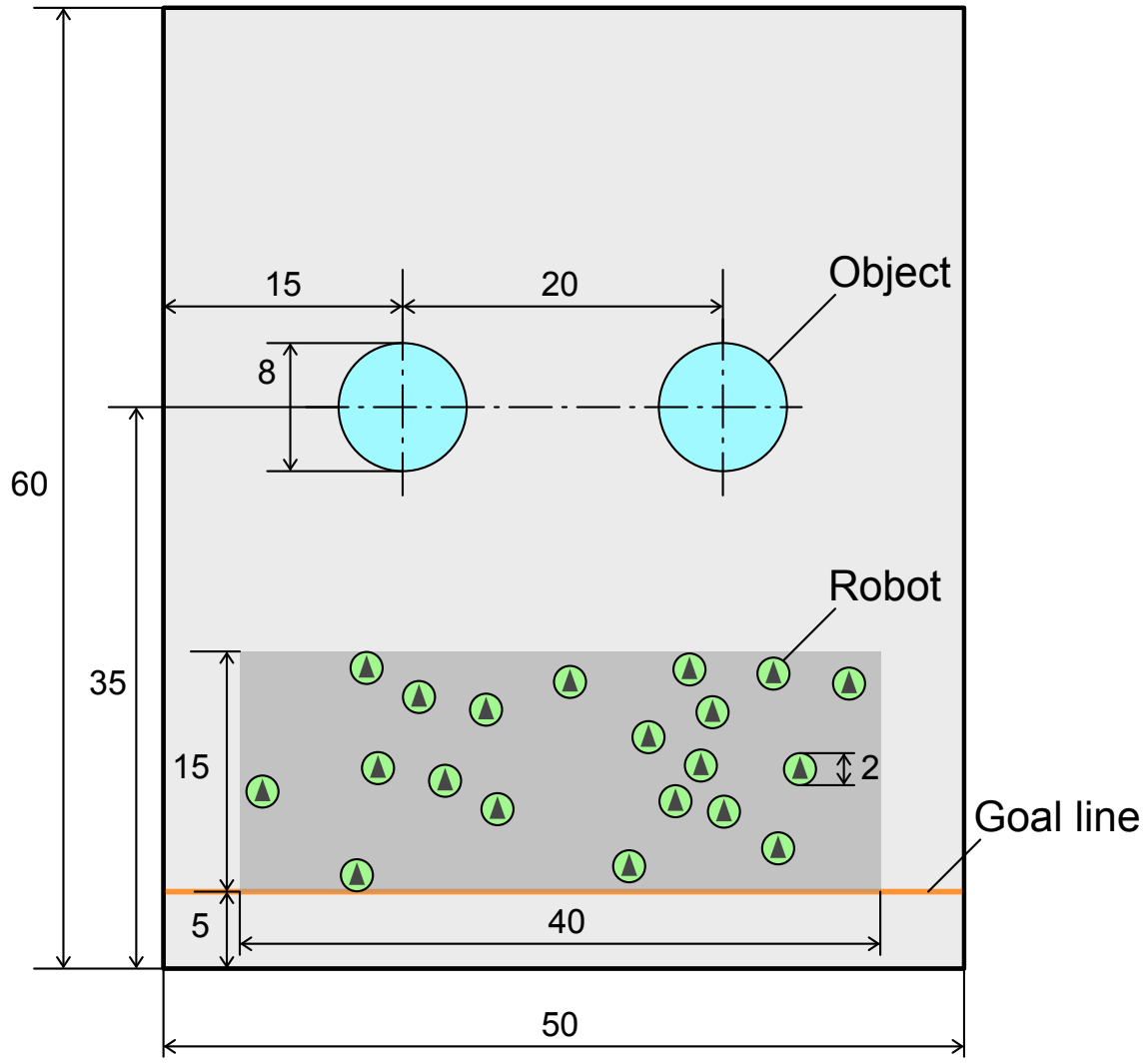


Fig. 7.3. Illustration of the task environment. The initial positions of the robots are randomly placed within the gray shaded area.

a value that corresponds to the distance to the detected item or returns 0 if there are no items within the sensor range. The sensor value from the distance sensor is normalized into a real value within the range of $[0, 1]$.

The electronic compass returns two real values corresponding to the facing direction of the robot, i.e., the sine and cosine of the robot direction. The values from the electronic compass are normalized in the range of $[0, 1]$.

Additionally, one bias input is introduced that always returns the value of 0.5. In total, eighteen inputs are collected from the sensors of the robot. The input values are fed into the robot controller and return the output values for controlling the left and right motors of the robot.

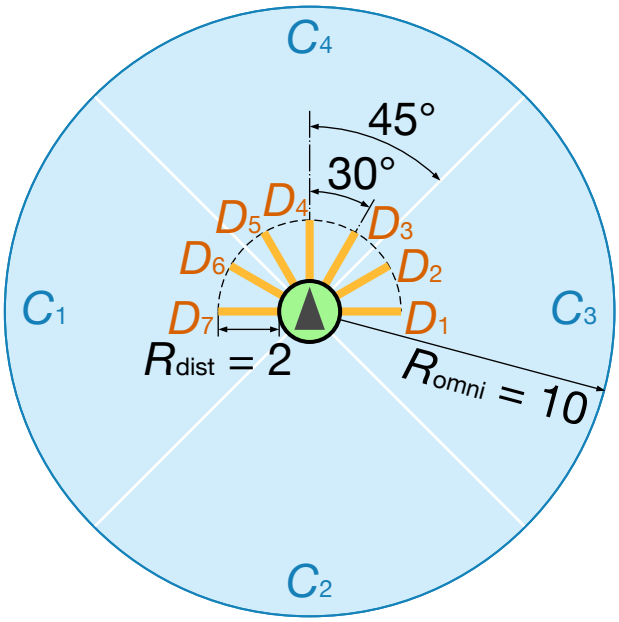


Fig. 7.4. Sensor settings of the robot. The R_{omni} and R_{dist} are the sensor range of the omnidirectional camera and the distance sensor, respectively. The circular-shaped sensor range of the omnidirectional camera is divided into four equal slices (C_1 to C_4). Distance sensors are attached to the front side of the robot with an angular interval of $\pi/6$ radians (D_1 to D_7).

7.2.3 Fitness Function

When designing controllers by the evolutionary robotics approach, an evolutionary algorithm evaluates and optimizes the robot controllers based on a predefined fitness function. The fitness function is typically designed based on the achievement of the task. The controllers with higher fitness values are selected to produce the next population of candidate controllers.

The fitness function for the cooperative transportation task is defined based on the transport distance of the objects. First, a population of λ controllers is initialized with having a minimal structure. The controller with a minimal structure has no hidden nodes, as shown in Fig. 7.5. Each controller is copied to twenty robots and performs the cooperative transportation task. The fitness value F of the controller is calculated using the following equation:

$$F = \sum_{i=1}^{2} \max(0, d_i) + 30.0c, \tag{7.6}$$

where d_i is the vertical distance of the ith object transported towards the goal line, and c is the number of objects that have reached the goal line. If the ith object has reached $d_i = 30.0$, it will be considered to have reached the goal line. When the object is transported

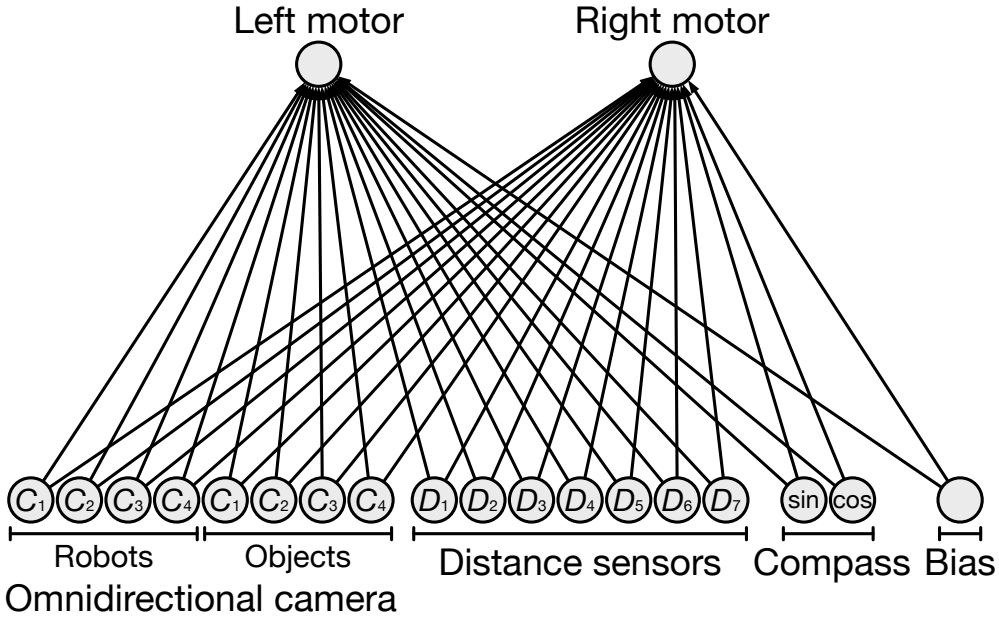


Fig. 7.5. Initial structure of the controller.

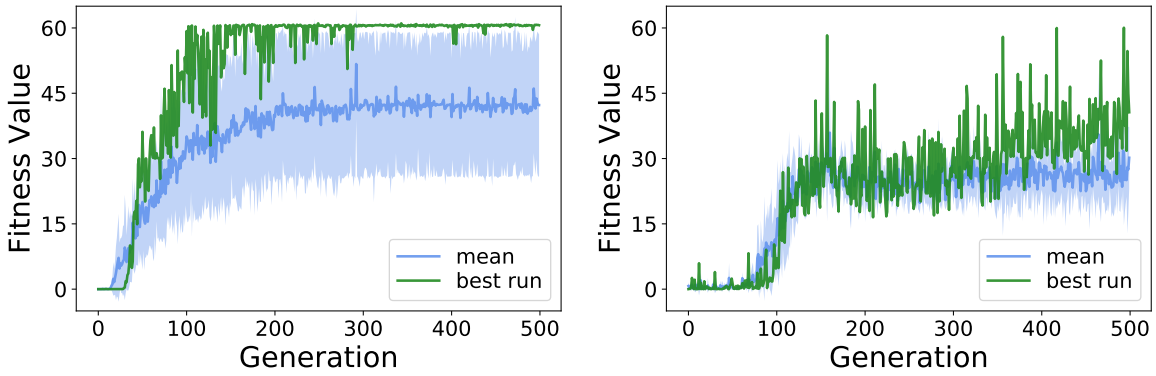
Table 7.1. Mutation probabilities of MBEANN.

Mutation probability	Value
Add-node mutation p_{node}	0.03
Add-connection mutation p_{link}	0.3
Synaptic weight mutation p_{weight}	1.0

to the goal line, a new object will be generated at the initial position. The fitness value is determined at the end of the task. The time limit of each simulation is set to 3000 time steps. The controllers with higher fitness values are selected to reproduce the next population. The population for the next generation is varied by using genetic operators. These processes are repeated until the maximum number of generations.

7.3 Results and Discussion

In this chapter, MBEANN and NEAT are used to design a controller for the robotic swarm. The mutation probabilities of the MBEANN are listed in Table 7.1. As for the selection method of the MBEANN approach, a tournament selection with a tournament size of 20 is applied. The NEAT algorithm is implemented using the NEAT-Python library [83] with the parameter settings determined based on the double pole balancing problem without velocities described in [106]. The weight values are initialized with random values sampled from a Gaussian distribution with a mean of 0 and a standard deviation of 0.05. Each weight



(a) MBEANN. **(b)** NEAT.

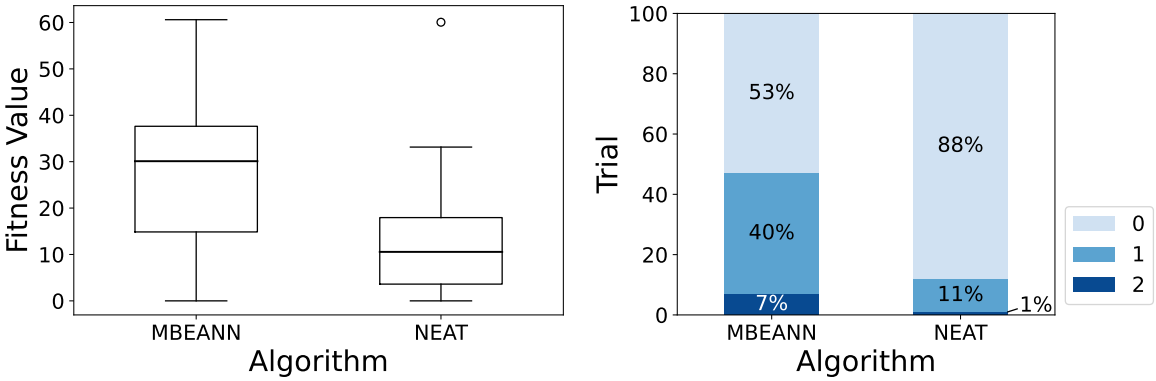
Fig. 7.6. Transitions of the highest fitness values. The "mean" indicates the mean value of the highest fitness values over the five evolutionary processes, while the shaded area corresponds to the standard deviation. The "best run" indicates the transition of the evolutionary process that has obtained the highest fitness value among the five evolutionary processes.

value takes a real number within the range $[-5, 5]$. The population size is set to $\lambda = 320$ and the maximum generation of 500.

Five independent evolutionary processes are executed with different random seeds. The transitions of the highest fitness values are shown in Fig. 7.6. In the case of MBEANN, two out of five evolutionary processes stagnate around the fitness value of 60, while the other three stagnate around 30. On the other hand, NEAT has scored lower fitness values where only one out of the five processes stagnates above 30, and the other four stagnate just below 30. Moreover, NEAT shows a slow increase in the fitness values for all five evolutionary processes scoring around 0 during the first 100 generations.

For further comparison, the controller that has obtained the highest fitness value among the five evolutionary processes is selected and re-evaluated by executing the task for 100 trials. In this re-evaluation, the time limit of the task is extended to 6000 time steps. The highest fitness value is obtained in the 343rd generation for MBEANN and the 493rd generation for NEAT. The results of the re-evaluation are shown in Fig. 7.7. When comparing the fitness values scored in the re-evaluation, MBEANN scored significantly higher values than NEAT (Mann-Whitney U test, p-value < 0.001). Moreover, the controller evolved using NEAT showed a low performance by failing to transport an object for 88 out of 100 trials (see also Fig. 7.7(b)). In contrast, the controller evolved with MBEANN succeeded in transporting at least one object for 47 trials.

The total number of nodes and connections included in the best-evolved controller is summarized in Table 7.2. The best-evolved controller in MBEANN has 38 hidden nodes,



(a) Comparison of the fitness values.

(b) Comparison of the number of objects that have reached the goal line.

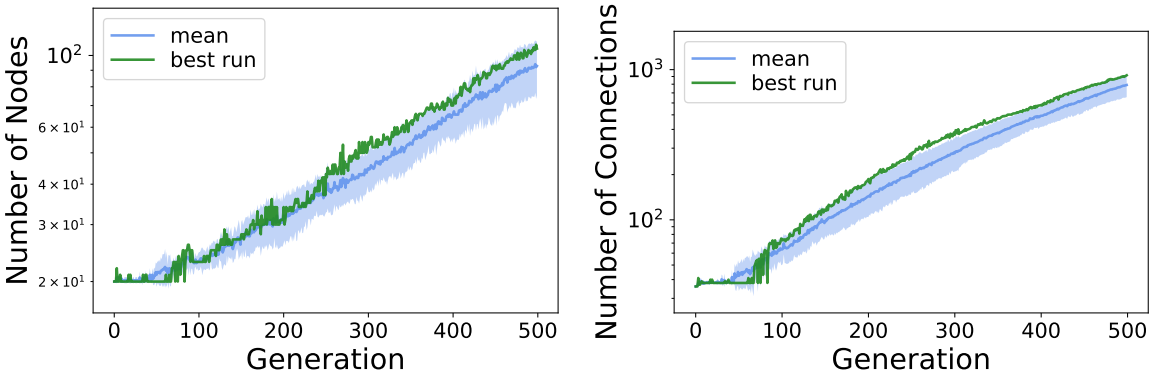
Fig. 7.7. Comparisons of the performance of the best-evolved controller developed with MBEANN and NEAT over 100 trials with different random seeds. Each trial lasts for 6000 time steps.

Table 7.2. Total numbers of nodes and connections of the best-evolved controller. Eighteen input nodes and two output nodes are included in the number of nodes.

	Operon ID	Number of nodes	Number of connections
MBEANN	0	20	38
	1	6	102
	2	10	84
	3	4	69
	4	6	51
	5	5	47
	6	6	36
	7	1	23
	total	58	450
NEAT		31	261

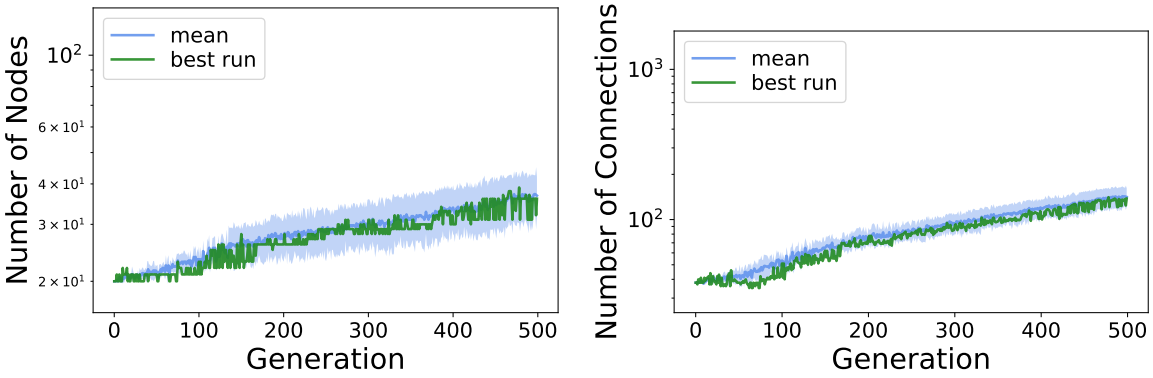
while NEAT has only 11 hidden nodes. Additionally, Figs. 7.8 and 7.9 show the topological structure of the controller that has obtained the highest fitness within each generation. The number of nodes grows exponentially in the evolutionary process using MBEANN, while NEAT increases linearly. However, due to the neutrality of the structural mutations, controllers developed with MBEANN showed higher performance even with large network structures.

The maximum number of available connections depends on the number of nodes, and



(a) Transitions of the number of nodes. **(b)** Transitions of the number of connections.

Fig. 7.8. Transitions of the topological structure of the controller that has obtained the highest fitness value within each generation using MBEANN. The "best run" correspond to the evolutionary processes in Fig. 7.6(a).



(a) Transitions of the number of nodes. **(b)** Transitions of the number of connections.

Fig. 7.9. Transitions of the topological structure of the controller that has obtained the highest fitness value within each generation using NEAT. The "best run" correspond to the evolutionary processes in Fig. 7.6(b).

hence the network size of the controller is more dominated by the number of nodes. In this chapter, the probability of the add-node mutation of both MBEANN and NEAT is set to 0.03. The MBEANN algorithm has a faster growth of the network structure because the add-node mutation is applied to each operon with a constant probability. This characteristic will be beneficial if the task to be performed requires a neural network with many nodes. On the other hand, the probability for the add-node mutation must be carefully set depending on the task to prevent the neural network from excessive bloating. Moreover, a neural network with a complex structure is typically not easy to analyze and interpret behavior.

7.4 Conclusions

This chapter applied the MBEANN approach to design a controller for a robotic swarm. MBEANN is a TWEANN algorithm that only applies mutations to evolve neural networks. The controller evolved with MBEANN was compared with the controller designed by NEAT. The controllers were evaluated in a cooperative transportation task, in which robots cooperate to transport heavy objects to the goal line. The results of the experiments showed that the controller evolved with MBEANN significantly outperformed the NEAT controller.

Chapter 8

Conclusions

This thesis presented automatic design methods for designing controllers for robotic swarms. One of the challenges in the field of swarm robotics is designing control software for a robotic swarm. The most common approach in the swarm robotics research community follows a trial and error process to design a controller. This method is effective if the desired collective behavior is simple enough for the designer to understand and program into the controller. However, this method is guided only by the designer's intuition and experience. As an alternative method, the automatic design method develops controllers by transforming the design problem into an optimization problem. This thesis focused on the evolutionary robotics approach, which is the most often used automatic design method. So far, there has been little progress in studies using automatic design methods within the swarm robotics community. This thesis contributed to the swarm robotics community from the following two aspects.

First, this thesis developed controllers for robotic swarms using the evolutionary robotics approach to perform tasks that are difficult to design controllers by hand. Chapter 3 showed how the evolutionary robotics approach could be applied to generate collective cognition by robotic swarms in the foraging task. In this chapter, the controller for the robotic swarm was developed to distinguish between two types of objects and transport one of them to the goal. The robot could not distinguish between the two types of objects because the sensory inputs were set to handle them as the same objects. The controller for the robotic swarm was successfully developed to accomplish the foraging task. In Chapter 4, the controller was developed to manage congestion in the path-formation task. The developed controller enabled the robots to perform behavioral specialization to mitigate congestion. In addition, Chapter 5 further investigated the behavioral specialization within the robotic swarm that emerged in Chapter 4. Chapter 5 showed the importance of the embodiment of robots which is sometimes neglected in swarm robotics. The robot embodiment would influence not only the performance but also the behavioral specialization to solve the task.

Second, this thesis presented novel evolutionary robotics approaches to design controllers for robotic swarms. Chapter 6 proposed an evolutionary robotics approach that applied echo state networks as robot controllers. The results showed that echo state networks could reduce the effort required in designing the parameters of the robot controller. Besides, the echo state networks showed a faster convergence in optimizing controllers than the traditional recurrent neural networks. Chapter 7 proposed an evolutionary robotics approach that evolves both the weight values and the structure of neural networks. The algorithm called Mutation-Based Evolving Artificial Neural Network (MBEANN) was employed to design the controller for the robotic swarm. The performance was compared with NeuroEvolution of Augmenting Topologies (NEAT), which is a widely used algorithm to evolve both topologies and weights. The results showed that the robot controller evolved with MBEANN outperformed the NEAT controller in the cooperative transportation task.

8.1 Future Work

Despite its potential to solve tasks, robotic swarms have yet to be adopted for real-world problems [21, 25]. Typically, the evolutionary robotics approach requires a considerable amount of time. Therefore, in this thesis, all of the experiments were conducted in computer simulations. When considering the experiments using robotic swarms in the real world, the major problem will be the *reality gap* [39, 59, 101]. The reality gap is the problem that occurs when controllers developed in computer simulations are transferred to the physical robots. In more detail, the controllers that showed high performance in computer simulations become ineffective when transferred to the robots in the real world.

As for now, there is no general method to overcome the reality gap problem. Therefore, overcoming the reality gap could be one of the possible future directions. Some recent promising approaches to reduce the reality gap, e.g., [67, 78], could be combined with the evolutionary robotics approaches described in the thesis. Another way to overcome the reality gap could be online evolution [11, 39], which evolves controllers while the robots perform their tasks. The development of a novel approach for online evolution is left for future work. In addition, further developments of the research on digital twins [42] and cyber-physical systems [86, 112] could be merged with swarm robotics to address real-world problems.

Furthermore, when considering real-world applications, it is important to know how robotic swarms are behaving. In particular, robotic swarms should avoid causing accidents or harm due to unexpected behaviors. The collective behavior is often considered as a complex system that is difficult to analyze and understand its behavior [84, 97]. Therefore, the research on a general analysis of collective behavior is left for future work. Also, neural networks are typically difficult to understand how they are working. Hence, robot controllers developed

with the evolutionary robotics approach using neural networks are difficult to understand their behavior. The research on neural networks is rapidly progressing with the help of the deep learning community. Further developments on neural networks might make it possible to understand the behavior of robotic swarms that use neural networks. The research on human-swarm interaction [65] could be another fundamental topic when considering real-world applications with a human operator taking control of robotic swarms. As for now, almost none of the studies have focused on human-swarm interaction using automatic design methods. Therefore, how to apply automatic design methods for human-swarm interaction is left as possible future work.

References

[1] Alkilabi, M. H. M., Narayan, A., and Tuci, E. Cooperative object transport with a swarm of e-puck robots: Robustness and scalability of evolved collective strategies. *Swarm Intelligence*, Vol. 11, No. 3-4, pp. 185–209, 2017.

[2] Baldassarre, G., Nolfi, S., and Parisi, D. Evolving mobile robots able to display collective behaviors. *Artificial Life*, Vol. 9, No. 3, pp. 255–267, 2003.

[3] Banzhaf, W., Nordin, P., Keller, R. E., and Francone, F. D. *Genetic Programming: An Introduction*. Morgan Kaufmann San Francisco, 1998.

[4] Bayındır, L. A review of swarm robotics tasks. *Neurocomputing*, Vol. 172, pp. 292–321, 2016.

[5] Beni, G. From swarm intelligence to swarm robotics. In Şahin, E. and Spears, W. M. (Eds.), *Swarm Robotics*, Vol. 3342 of *Lecture Notes in Computer Science*, pp. 1–9. Springer, 2005.

[6] Beyer, H.-G. Evolution strategies. *Scholarpedia*, Vol. 2, No. 8, p. 1965, 2007.

[7] Beyer, H.-G. and Schwefel, H.-P. Evolution strategies: A comprehensive introduction. *Natural Computing*, Vol. 1, No. 1, pp. 3–52, 2002.

[8] Birattari, M., Ligot, A., Bozhinoski, D., Brambilla, M., Francesca, G., Garattoni, L., Garzón Ramos, D., Hasselmann, K., Kegeleirs, M., Kuckling, J., Pagnozzi, F., Roli, A., Salman, M., and Stützle, T. Automatic off-line design of robot swarms: A manifesto. *Frontiers in Robotics and AI*, Vol. 6, No. 59, 2019.

[9] Bonabeau, E., Dorigo, M., and Theraulaz, G. *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press, 1999.

[10] Brambilla, M., Ferrante, E., Birattari, M., and Dorigo, M. Swarm robotics: A review from the swarm engineering perspective. *Swarm Intelligence*, Vol. 7, No. 1, pp. 1–41, 2013.

[11] Bredeche, N., Haasdijk, E., and Eiben, A. On-line, on-board evolution of robot controllers. In Collet, P., Monmarché, N., Legrand, P., Schoenauer, M., and Lutton, E. (Eds.), *Artifical Evolution*, Vol. 5975 of *Lecture Notes in Computer Science*, pp. 110–121. Springer, 2010.

[12] Brutschy, A., Pini, G., Pinciroli, C., Birattari, M., and Dorigo, M. Self-organized task

allocation to sequentially interdependent tasks in swarm robotics. *Autonomous Agents and Multi-Agent Systems*, Vol. 28, No. 1, pp. 101–125, 2014.

[13] Cabinet Office, Government of Japan. Society 5.0. `https://www8.cao.go.jp/cstp/english/society5_0/index.html`.

[14] Camazine, S., Deneubourg, J.-L., Franks, N. R., Sneyd, J., Theraula, G., and Bonabeau, E. *Self-Organization in Biological Systems*. Princeton University Press, 2003.

[15] Catto, E. Box2D: A 2D physics engine for games. Available at `https://box2d.org`.

[16] Cliff, D., Husbands, P., and Harvey, I. Explorations in evolutionary robotics. *Adaptive Behavior*, Vol. 2, No. 1, pp. 73–110, 1993.

[17] Couzin, I. D. Collective cognition in animal groups. *Trends in Cognitive Sciences*, Vol. 13, No. 1, pp. 36–43, 2009.

[18] Darwin, C. *On the Origin of Species by Means of Natural Selection or the Preservation of Favoured Races in the Struggle for Life*. John Murray, 1859.

[19] Davies, N. B., Krebs, J. R., and West, S. A. *An Introduction to Behavioural Ecology*. John Wiley & Sons, 2012.

[20] Dorigo, M. Ant colony optimization. *Scholarpedia*, Vol. 2, No. 3, p. 1461, 2007.

[21] Dorigo, M. and Birattari, M. Swarm intelligence. *Scholarpedia*, Vol. 2, No. 9, p. 1462, 2007.

[22] Dorigo, M., Birattari, M., and Brambilla, M. Swarm robotics. *Scholarpedia*, Vol. 9, No. 1, p. 1463, 2014.

[23] Dorigo, M., Birattari, M., and Stutzle, T. Ant colony optimization. *IEEE Computational Intelligence Magazine*, Vol. 1, No. 4, pp. 28–39, 2006.

[24] Dorigo, M., Maniezzo, V., and Colorni, A. Ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, Vol. 26, No. 1, pp. 29–41, 1996.

[25] Dorigo, M., Theraulaz, G., and Trianni, V. Swarm robotics: Past, present, and future. *Proceedings of the IEEE*, Vol. 109, No. 7, pp. 1152–1165, 2021.

[26] Duarte, M., Costa, V., Gomes, J., Rodrigues, T., Silva, F., Oliveira, S. M., and Christensen, A. L. Evolution of collective behaviors for a real swarm of aquatic surface robots. *PLOS ONE*, Vol. 11, No. 3, 2016.

[27] Dussutour, A., Beshers, S., Deneubourg, J.-L., and Fourcassié, V. Priority rules govern the organization of traffic on foraging trails under crowding conditions in the leaf-cutting ant *Atta colombica*. *Journal of Experimental Biology*, Vol. 212, No. 4, pp. 499–505, 2009.

[28] Eberhart, R. and Kennedy, J. A new optimizer using particle swarm theory. In *Proceedings of the 6th International Symposium on Micro Machine and Human Science*, pp. 39–43. IEEE, 1995.

[29] Ebert, J. T., Gauci, M., and Nagpal, R. Multi-feature collective decision making in robot

swarms. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pp. 1711–1719. International Foundation for Autonomous Agents and Multiagent Systems, 2018.

[30] Eiben, A. E. and Smith, J. E. *Introduction to Evolutionary Computing*. Springer, 2003.

[31] Engelbrecht, A. P. *Computational Intelligence: An Introduction*. John Wiley & Sons, 2007.

[32] Ferrante, E., Turgut, A. E., Duéñez-Guzmán, E., Dorigo, M., and Wenseleers, T. Evolution of self-organized task specialization in robot swarms. *PLOS Computational Biology*, Vol. 11, No. 8, 2015.

[33] Floreano, D., Dürr, P., and Mattiussi, C. Neuroevolution: From architectures to learning. *Evolutionary Intelligence*, Vol. 1, No. 1, pp. 47–62, 2008.

[34] Floreano, D. and Mondada, F. Automatic creation of an autonomous agent: Genetic evolution of a neural network driven robot. In Cliff, D., Husbands, P., Meyer, J.-A., and Wilson, S. W. (Eds.), *From Animals to Animats 3*, pp. 421–430. MIT Press, 1994.

[35] Fogel, D. B. and Fogel, L. J. An introduction to evolutionary programming. In Alliot, J.-M., Lutton, E., Ronald, E., Schoenauer, M., and Snyers, D. (Eds.), *Artificial Evolution*, Vol. 1063 of *Lecture Notes in Computer Science*, pp. 21–33. Springer, 1996.

[36] Fogel, L., Owens, A., and Walsh, M. *Artificial Intelligence Through Simulated Evolution*. John Wiley & Sons, 1966.

[37] Fogel, L. J. Autonomous automata. *Industrial Research*, Vol. 4, pp. 14–19, 1962.

[38] Fourcassié, V., Dussutour, A., and Deneubourg, J.-L. Ant traffic rules. *Journal of Experimental Biology*, Vol. 213, No. 14, pp. 2357–2363, 2010.

[39] Francesca, G. and Birattari, M. Automatic design of robot swarms: Achievements and challenges. *Frontiers in Robotics and AI*, Vol. 3, No. 29, 2016.

[40] Francesca, G., Brambilla, M., Brutschy, A., Trianni, V., and Birattari, M. AutoMoDe: A novel approach to the automatic design of control software for robot swarms. *Swarm Intelligence*, Vol. 8, No. 2, pp. 89–112, 2014.

[41] Fukuyama, M. Society 5.0: Aiming for a new human-centered society. *Japan SPOTLIGHT*, Vol. 37, No. 4, pp. 47–50, 2018.

[42] Fuller, A., Fan, Z., Day, C., and Barlow, C. Digital twin: Enabling technologies, challenges and open research. *IEEE Access*, Vol. 8, pp. 108952–108971, 2020.

[43] Goldberg, D. E. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, 1989.

[44] Gomes, J., Urbano, P., and Christensen, A. L. Evolution of swarm robotics systems with novelty search. *Swarm Intelligence*, Vol. 7, No. 2-3, pp. 115–144, 2013.

[45] Goodfellow, I., Bengio, Y., and Courville, A. *Deep Learning*. MIT Press, 2016.

[46] Groß, R. and Dorigo, M. Towards group transport by swarms of robots. *International Journal of Bio-Inspired Computation*, Vol. 1, No. 1-2, pp. 1–13, 2009.

[47] Hamann, H. Towards swarm calculus: Urn models of collective decisions and universal properties of swarm performance. *Swarm Intelligence*, Vol. 7, No. 2-3, pp. 145–172, 2013.

[48] Hamann, H. *Swarm Robotics: A Formal Approach*. Springer, 2018.

[49] Hansen, N. The CMA evolution strategy: A tutorial. *arXiv preprint arXiv:1604.00772*, 2016.

[50] Hansen, N. and Ostermeier, A. Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation. In *Proceedings of the IEEE International Conference on Evolutionary Computation*, pp. 312–317. 1996.

[51] Hansen, N. and Ostermeier, A. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, Vol. 9, No. 2, pp. 159–195, 2001.

[52] Harvey, I., Husbands, P., and Cliff, D. Issues in evolutionary robotics. In Meyer, J.-A., Roitblat, H. L., and Wilson, S. W. (Eds.), *From Animals to Animats 2*, pp. 364–373. MIT Press, 1993.

[53] Hiraga, M. pyMBEANN. Available at `https://github.com/motoHiraga/pyMBEANN`.

[54] Hiraga, M., Wei, Y., Yasuda, T., and Ohkura, K. Evolving autonomous specialization in congested path formation task of robotic swarms. *Artificial Life and Robotics*, Vol. 23, No. 4, pp. 547–554, 2018.

[55] Holland, J. H. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. MIT Press, 1992.

[56] Hüttenrauch, M., Šošić, A., and Neumann, G. Deep reinforcement learning for swarm systems. *Journal of Machine Learning Research*, Vol. 20, No. 54, pp. 1–31, 2019.

[57] Jaeger, H. The "echo state" approach to analysing and training recurrent neural networks. *Tech. Rep. 148*, German National Research Center for Information Technology (GMD), 2001.

[58] Jaeger, H. Echo state network. *Scholarpedia*, Vol. 2, No. 9, p. 2330, 2007.

[59] Jakobi, N., Husbands, P., and Harvey, I. Noise and the reality gap: The use of simulation in evolutionary robotics. In Morán, F., Moreno, A., Merelo, J. J., and Chacón, P. (Eds.), *Advances in Artificial Life*, Vol. 929 of *Lecture Notes in Computer Science*, pp. 704–720. Springer, 1995.

[60] Jin, B., Liang, Y., Han, Z., Hiraga, M., and Ohkura, K. A hierarchical training method of generating collective foraging behavior for a robotic swarm. *Artificial Life and Robotics*, Vol. 27, No. 1, pp. 137–141, 2022.

[61] Karaboga, D. An idea based on honey bee swarm for numerical optimization. *Tech. Rep. TR06*, Erciyes University, Engineering Faculty, Computer Engineering Department, 2005.

[62] Karaboga, D. and Basturk, B. A powerful and efficient algorithm for numerical function optimization: Artificial bee colony (ABC) algorithm. *Journal of Global Optimization*,

Vol. 39, No. 3, pp. 459–471, 2007.

[63] Karaboga, D. and Basturk, B. On the performance of artificial bee colony (ABC) algorithm. *Applied Soft Computing*, Vol. 8, No. 1, pp. 687–697, 2008.

[64] Kennedy, J. and Eberhart, R. Particle swarm optimization. In *Proceedings of the IEEE International Conference on Neural Networks*, Vol. 4, pp. 1942–1948. IEEE, 1995.

[65] Kolling, A., Walker, P., Chakraborty, N., Sycara, K., and Lewis, M. Human interaction with robot swarms: A survey. *IEEE Transactions on Human-Machine Systems*, Vol. 46, No. 1, pp. 9–26, 2015.

[66] König, L., Mostaghim, S., and Schmeck, H. Decentralized evolution of robotic behavior using finite state machines. *International Journal of Intelligent Computing and Cybernetics*, Vol. 2, No. 4, pp. 695–723, 2009.

[67] Koos, S., Mouret, J.-B., and Doncieux, S. The transferability approach: Crossing the reality gap in evolutionary robotics. *IEEE Transactions on Evolutionary Computation*, Vol. 17, No. 1, pp. 122–145, 2012.

[68] Koza, J. R. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, Vol. 1. MIT Press, 1992.

[69] Koza, J. R. *Genetic Programming II: Automatic Discovery of Reusable Programs*, Vol. 2. MIT Press, 1994.

[70] Krieger, M. J. and Billeter, J.-B. The call of duty: Self-organised task allocation in a population of up to twelve mobile robots. *Robotics and Autonomous Systems*, Vol. 30, No. 1-2, pp. 65–84, 2000.

[71] Labella, T. H., Dorigo, M., and Deneubourg, J.-L. Division of labor in a group of robots inspired by ants' foraging behavior. *ACM Transactions on Autonomous and Adaptive Systems*, Vol. 1, No. 1, pp. 4–25, 2006.

[72] Lauer, K. pybox2d: 2D game physics for Python. Available at `https://github.com/pybox2d/pybox2d`.

[73] LeCun, Y., Bengio, Y., and Hinton, G. Deep learning. *Nature*, Vol. 521, pp. 436–444, 2015.

[74] Lehman, J. and Miikkulainen, R. Neuroevolution. *Scholarpedia*, Vol. 8, No. 6, p. 30977, 2013.

[75] Lerman, K. and Galstyan, A. Mathematical model of foraging in a group of robots: Effect of interference. *Autonomous Robots*, Vol. 13, No. 2, pp. 127–141, 2002.

[76] Lewis, M. A., Fagg, A. H., and Solidum, A. Genetic programming approach to the construction of a neural network for control of a walking robot. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 2618–2623. 1992.

[77] Li, S., Batra, R., Brown, D., Chang, H.-D., Ranganathan, N., Hoberman, C., Rus, D., and Lipson, H. Particle robotics based on statistical mechanics of loosely coupled components. *Nature*, Vol. 567, pp. 361–365, 2019.

[78] Ligot, A. and Birattari, M. Simulation-only experiments to mimic the effects of the reality gap in the automatic design of robot swarms. *Swarm Intelligence*, Vol. 14, No. 1, pp. 1–24, 2020.

[79] Liu, W., Winfield, A. F., Sa, J., Chen, J., and Dou, L. Towards energy optimization: Emergent task allocation in a swarm of foraging robots. *Adaptive Behavior*, Vol. 15, No. 3, pp. 289–305, 2007.

[80] Lukoševičius, M. and Jaeger, H. Reservoir computing approaches to recurrent neural network training. *Computer Science Review*, Vol. 3, No. 3, pp. 127–149, 2009.

[81] Maass, W., Natschläger, T., and Markram, H. Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural Computation*, Vol. 14, No. 11, pp. 2531–2560, 2002.

[82] Marcolino, L. S., dos Passos, Y. T., de Souza, Á. A. F., dos Santos Rodrigues, A., and Chaimowicz, L. Avoiding target congestion on the navigation of robotic swarms. *Autonomous Robots*, Vol. 41, No. 6, pp. 1297–1320, 2017.

[83] McIntyre, A., Kallada, M., Miguel, C. G., and da Silva, C. F. NEAT-Python. Available at `https://github.com/CodeReclaimers/neat-python`.

[84] Mitchell, M. *Complexity: A Guided Tour*. Oxford University Press, 2009.

[85] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. Human-level control through deep reinforcement learning. *Nature*, Vol. 518, pp. 529–533, 2015.

[86] Monostori, L., Kádár, B., Bauernhansl, T., Kondoh, S., Kumara, S., Reinhart, G., Sauer, O., Schuh, G., Sihn, W., and Ueda, K. Cyber-physical systems in manufacturing. *CIRP Annals*, Vol. 65, No. 2, pp. 621–641, 2016.

[87] Morimoto, D., Hiraga, M., Ohkura, K., and Matsumura, Y. Generating and analyzing collective behavior in a robotic swarm by the use of deep reinforcement learning and deep neuroevolution. *Transactions of the Institute of Systems, Control and Information Engineers*, Vol. 33, No. 5, pp. 163–170, 2020 (in Japanese).

[88] Nelson, A. L., Barlow, G. J., and Doitsidis, L. Fitness functions in evolutionary robotics: A survey and analysis. *Robotics and Autonomous Systems*, Vol. 57, No. 4, pp. 345–370, 2009.

[89] Nguyen, T. and Banerjee, B. Reinforcement learning as a rehearsal for swarm foraging. *Swarm Intelligence*, Vol. 16, No. 1, pp. 29–58, 2022.

[90] Nolfi, S. and Floreano, D. *Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines*. MIT Press, 2000.

[91] Ohkura, K., Yasuda, T., Kawamatsu, Y., Matsumura, Y., and Ueda, K. MBEANN: Mutation-based evolving artificial neural networks. In Almeida e Costa, F., Rocha,

L. M., Costa, E., Harvey, I., and Coutinho, A. (Eds.), *Advances in Artificial Life*, Vol. 4648 of *Lecture Notes in Computer Science*, pp. 936–945. Springer, 2007.

[92] Panait, L. and Luke, S. Cooperative multi-agent learning: The state of the art. *Autonomous Agents and Multi-Agent Systems*, Vol. 11, pp. 387–434, 2005.

[93] Poli, R., Kennedy, J., and Blackwell, T. Particle swarm optimization: An overview. *Swarm Intelligence*, Vol. 1, No. 1, pp. 33–57, 2007.

[94] Price, K., Storn, R. M., and Lampinen, J. A. *Differential Evolution: A Practical Approach to Global Optimization*. Natural Computing Series. Springer, 2005.

[95] Rubenstein, M., Cornejo, A., and Nagpal, R. Programmable self-assembly in a thousand-robot swarm. *Science*, Vol. 345, No. 6198, pp. 795–799, 2014.

[96] Şahin, E. Swarm robotics: From sources of inspiration to domains of application. In Şahin, E. and Spears, W. M. (Eds.), *Swarm Robotics*, Vol. 3342 of *Lecture Notes in Computer Science*, pp. 10–20. Springer, 2005.

[97] Sayama, H. *Introduction to the Modeling and Analysis of Complex Systems*. Open SUNY Textbooks, 2015.

[98] Schmickl, T., Möslinger, C., and Crailsheim, K. Collective perception in a robot swarm. In Şahin, E., Spears, W. M., and Winfield, A. F. T. (Eds.), *Swarm Robotics*, Vol. 4433 of *Lecture Notes in Computer Science*, pp. 144–157. Springer, 2007.

[99] Schwefel, H.-P. *Numerical Optimization of Computer Modelsodels*. John Wiley & Sons, 1981.

[100] Schwefel, H.-P. *Evolution and Optimum Seeking*. John Wiley & Sons, 1995.

[101] Silva, F., Duarte, M., Correia, L., Oliveira, S. M., and Christensen, A. L. Open issues in evolutionary robotics. *Evolutionary Computation*, Vol. 24, No. 2, pp. 205–236, 2016.

[102] Silva, F., Urbano, P., Correia, L., and Christensen, A. L. odNEAT: An algorithm for decentralised online evolution of robotic controllers. *Evolutionary Computation*, Vol. 23, No. 3, pp. 421–449, 2015.

[103] Soysal, O., Bahçeci, E., and Şahin, E. Aggregation in swarm robotic systems: Evolution and probabilistic control. *Turkish Journal of Electrical Engineering & Computer Sciences*, Vol. 15, No. 2, pp. 199–225, 2007.

[104] Sperati, V., Trianni, V., and Nolfi, S. Self-organised path formation in a swarm of robots. *Swarm Intelligence*, Vol. 5, No. 2, pp. 97–119, 2011.

[105] Stanley, K. O., Clune, J., Lehman, J., and Miikkulainen, R. Designing neural networks through neuroevolution. *Nature Machine Intelligence*, Vol. 1, No. 1, pp. 24–35, 2019.

[106] Stanley, K. O. and Miikkulainen, R. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, Vol. 10, No. 2, pp. 99–127, 2002.

[107] Stone, P. and Veloso, M. Multiagent systems: A survey from a machine learning perspective. *Autonomous Robots*, Vol. 8, No. 3, pp. 345–383, 2000.

[108] Storn, R. and Price, K. Differential evolution: A simple and efficient heuristic for

global optimization over continuous spaces. *Journal of Global Optimization*, Vol. 11, No. 4, pp. 341–359, 1997.

[109] Such, F. P., Madhavan, V., Conti, E., Lehman, J., Stanley, K. O., and Clune, J. Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning. *arXiv preprint arXiv:1712.06567*, 2017.

[110] Sumpter, D. J. *Collective Animal Behavior*. Princeton University Press, 2010.

[111] Sutton, R. S. and Barto, A. G. *Reinforcement Learning: An Introduction*. MIT Press, 2018.

[112] Tao, F., Qi, Q., Wang, L., and Nee, A. Digital twins and cyber–physical systems toward smart manufacturing and industry 4.0: Correlation and comparison. *Engineering*, Vol. 5, No. 4, pp. 653–661, 2019.

[113] Trianni, V. *Evolutionary Swarm Robotics: Evolving Self-Organising Behaviours in Groups of Autonomous Robots*, Vol. 108 of *Studies in Computational Intelligence*. Springer, 2008.

[114] Trianni, V., Groß, R., Labella, T. H., Şahin, E., and Dorigo, M. Evolving aggregation behaviors in a swarm of robots. In Banzhaf, W., Ziegler, J., Christaller, T., Dittrich, P., and Kim, J. T. (Eds.), *Advances in Artificial Life*, Vol. 2801 of *Lecture Notes in Computer Science*, pp. 865–874. Springer, 2003.

[115] Trianni, V., Tuci, E., Passino, K. M., and Marshall, J. A. Swarm cognition: An interdisciplinary approach to the study of self-organising biological collectives. *Swarm Intelligence*, Vol. 5, No. 1, pp. 3–18, 2011.

[116] Tuci, E., Alkilabi, M. H. M., and Akanyeti, O. Cooperative object transport in multi-robot systems: A review of the state-of-the-art. *Frontiers in Robotics and AI*, Vol. 5, No. 59, 2018.

[117] Valentini, G., Brambilla, D., Hamann, H., and Dorigo, M. Collective perception of environmental features in a robot swarm. In Dorigo, M., Birattari, M., Li, X., López-Ibáñez, M., Ohkura, K., Pinciroli, C., and Stützle, T. (Eds.), *Swarm Intelligence*, Vol. 9882 of *Lecture Notes in Computer Science*, pp. 65–76. Springer, 2016.

[118] Vicsek, T. and Zafeiris, A. Collective motion. *Physics Reports*, Vol. 517, No. 3-4, pp. 71–140, 2012.

[119] Watson, R. A., Ficici, S. G., and Pollack, J. B. Embodied evolution: Distributing an evolutionary algorithm in a population of robots. *Robotics and Autonomous Systems*, Vol. 39, No. 1, pp. 1–18, 2002.

[120] Wei, Y., Hiraga, M., Ohkura, K., and Car, Z. Autonomous task allocation by artificial evolution for robotic swarms in complex tasks. *Artificial Life and Robotics*, Vol. 24, No. 1, pp. 127–134, 2019.

[121] Wei, Y., Nie, X., Hiraga, M., Ohkura, K., and Car, Z. Developing end-to-end control policies for robotic swarms using deep Q-learning. *Journal of Advanced Computational*

Intelligence and Intelligent Informatics, Vol. 23, No. 5, pp. 920–927, 2019.

[122] Yao, X. Evolving artificial neural networks. *Proceedings of the IEEE*, Vol. 87, No. 9, pp. 1423–1447, 1999.

[123] Yasuda, T. and Ohkura, K. Sharing experience for behavior generation of real swarm robot systems using deep reinforcement learning. *Journal of Robotics and Mechatronics*, Vol. 31, No. 4, pp. 520–525, 2019.

Appendix A

Publications Presented in the Thesis

This appendix provides a list of publications that are presented in the thesis. This appendix only shows a list of work published as the first author in academic journals and international conferences. The full list of publications is in Appendix B.

Chapter 3

- Motoaki Hiraga, Yufei Wei, and Kazuhiro Ohkura. Evolving collective cognition for object identification in foraging robotic swarms. *Artificial Life and Robotics*, Vol. 26, No. 1, pp. 21–28, 2021.
- Motoaki Hiraga, Yufei Wei, and Kazuhiro Ohkura. Evolving collective cognition of robotic swarms in the foraging task with poison. In *Proceedings of the 2019 IEEE Congress on Evolutionary Computation*, pp. 3205–3212, 2019.

Chapter 4

- Motoaki Hiraga, Toshiyuki Yasuda, and Kazuhiro Ohkura. Evolutionary acquisition of autonomous specialization in a path-formation task of a robotic swarm. *Journal of Advanced Computational Intelligence and Intelligent Informatics*, Vol. 22, No. 5, pp. 621–628, 2018.
- Motoaki Hiraga, Yufei Wei, Toshiyuki Yasuda, and Kazuhiro Ohkura. Evolving autonomous specialization in congested path formation task of robotic swarms. *Artificial Life and Robotics*, Vol. 23, No. 4, pp. 547–554, 2018.
- Motoaki Hiraga, Toshiyuki Yasuda, and Kazuhiro Ohkura. Evolutionary acquisition of congestion management of a robotic swarm in a path formation task. In *Proceedings of the 2nd International Symposium on Swarm Behavior and Bio-Inspired Robotics*,

pp. 141–146, 2017.

- Motoaki Hiraga and Kazuhiro Ohkura. Evolutionary emergence of path formation with autonomous specialization in a robotic swarm. In *Proceedings of the 2018 Conference on Artificial Life*, pp. 526–527, 2018.

Chapter 5

- Motoaki Hiraga and Kazuhiro Ohkura. Effects of congestion on swarm performance and autonomous specialization in robotic swarms. *Journal of Robotics and Mechatronics*, Vol. 31, No. 4, pp. 526–534, 2019.
- Motoaki Hiraga, Yasumasa Tamura, and Kazuhiro Ohkura. Behavioral specialization emerges from the embodiment of a robotic swarm. *Artificial Life and Robotics*, Vol. 25, No. 4, pp. 495–502, 2020.
- Motoaki Hiraga and Kazuhiro Ohkura. Effects of body size on autonomous specialization and congestion of robotic swarms. In *Proceedings of the 22nd Asia Pacific Symposium on Intelligent and Evolutionary Systems*, pp. 85–92, 2018.
- Motoaki Hiraga, Yasumasa Tamura, and Kazuhiro Ohkura. Effects of robot collisions on collective behavior in evolutionary robotic swarms. In *Proceedings of the 3rd International Symposium on Swarm Behavior and Bio-Inspired Robotics*, pp. 303–310, 2019.

Chapter 6

- Motoaki Hiraga, Yasumasa Tamura, and Kazuhiro Ohkura. Evolving echo state networks for generating collective behavior of a robotic swarm. In *Proceedings of the 4th International Symposium on Swarm Behavior and Bio-Inspired Robotics*, pp. 568–577, 2021.

Chapter 7

- Motoaki Hiraga and Kazuhiro Ohkura. Topology and weight evolving artificial neural networks in cooperative transport by a robotic swarm. *Artificial Life and Robotics*, accepted.
- Motoaki Hiraga and Kazuhiro Ohkura. MBEANN approach for evolving cooperative transport by a robotic swarm. In *Proceedings of the 4th International Symposium on Swarm Behavior and Bio-Inspired Robotics*, pp. 578–589, 2021.

Appendix B

List of Publications

Review Article

1. 大倉和博, 平賀元彰. ロボティックスワームにおける群れ行動生成手法：計算知能アプローチによる自動的設計. 計測と制御, Vol. 59, No. 2, pp. 131–136, 2020.

Journal Publications

1. 平賀元彰, 渡辺優, 大倉和博. 二重倒立振子制御問題への TWEANN アプローチ：NEAT と MBEANN の特性比較. システム制御情報学会論文誌, accepted.
2. Daichi Morimoto, Motoaki Hiraga, Naoya Shiozaki, Kazuhiro Ohkura, and Masaharu Munetomo. Evolving collective step-climbing behavior in multi-legged robotic swarm. *Artificial Life and Robotics*, accepted. https://doi.org/10.1007/s10015-021-00725-8.
3. Motoaki Hiraga and Kazuhiro Ohkura. Topology and weight evolving artificial neural networks in cooperative transport by a robotic swarm. *Artificial Life and Robotics*, accepted. https://doi.org/10.1007/s10015-021-00716-9.
4. Boyin Jin, Yupeng Liang, Ziyao Han, Motoaki Hiraga, and Kazuhiro Ohkura. A hierarchical training method of generating collective foraging behavior for a robotic swarm. *Artificial Life and Robotics*, Vol. 27, No. 1, pp. 137–141, 2022.
5. Motoaki Hiraga, Yufei Wei, and Kazuhiro Ohkura. Evolving collective cognition for object identification in foraging robotic swarms. *Artificial Life and Robotics*, Vol. 26, No. 1, pp. 21–28, 2021.
6. Motoaki Hiraga, Yasumasa Tamura, and Kazuhiro Ohkura. Behavioral specialization emerges from the embodiment of a robotic swarm. *Artificial Life and Robotics*, Vol. 25, No. 4, pp. 495–502, 2020.
7. 森本大智, 平賀元彰, 大倉和博, 松村嘉之. 深層強化学習と Deep Neuroevolution によるロボ

ティックスワームの群れ行動生成と解析. システム制御情報学会論文誌, Vol. 33, No. 5, pp. 163–170, 2020.

8. Yufei Wei, Xiaotong Nie, Motoaki Hiraga, Kazuhiro Ohkura, and Zlatan Car. Developing end-to-end control policies for robotic swarms using deep Q-learning. *Journal of Advanced Computational Intelligence and Intelligent Informatics*, Vol. 23, No. 5, pp. 920–927, 2019.
9. Motoaki Hiraga and Kazuhiro Ohkura. Effects of congestion on swarm performance and autonomous specialization in robotic swarms. *Journal of Robotics and Mechatronics*, Vol. 31, No. 4, pp. 526–534, 2019.
10. Yufei Wei, Motoaki Hiraga, Kazuhiro Ohkura, and Zlatan Car. Autonomous task allocation by artificial evolution for robotic swarms in complex tasks. *Artificial Life and Robotics*, Vol. 24, No. 1, pp. 127–134, 2019.
11. Motoaki Hiraga, Yufei Wei, Toshiyuki Yasuda, and Kazuhiro Ohkura. Evolving autonomous specialization in congested path formation task of robotic swarms. *Artificial Life and Robotics*, Vol. 23, No. 4, pp. 547–554, 2018.
12. Motoaki Hiraga, Toshiyuki Yasuda, and Kazuhiro Ohkura. Evolutionary acquisition of autonomous specialization in a path-formation task of a robotic swarm. *Journal of Advanced Computational Intelligence and Intelligent Informatics*, Vol. 22, No. 5, pp. 621–628, 2018.

International Conferences

1. Daichi Morimoto, Motoaki Hiraga, Naoya Shiozaki, Kazuhiro Ohkura, and Masaharu Munetomo. Evolutionary acquisition of collective behavior for a multi-legged robotic swarm. In *Proceedings of the 5th International Symposium on Swarm Behavior and Bio-Inspired Robotics*, pp. 1843–1848, 2022.
2. Motoaki Hiraga, Yasumasa Tamura, and Kazuhiro Ohkura. Evolving echo state networks for generating collective behavior of a robotic swarm. In *Proceedings of the 4th International Symposium on Swarm Behavior and Bio-Inspired Robotics*, pp. 568–577, 2021.
3. Motoaki Hiraga and Kazuhiro Ohkura. MBEANN approach for evolving cooperative transport by a robotic swarm. In *Proceedings of the 4th International Symposium on Swarm Behavior and Bio-Inspired Robotics*, pp. 578–589, 2021.
4. Daichi Morimoto, Motoaki Hiraga, Kazuhiro Ohkura, and Masaharu Munetomo. Generating collective step-climbing behavior using a multi-legged robotic swarm. In *Proceedings of the 4th International Symposium on Swarm Behavior and Bio-Inspired Robotics*, pp. 590–601, 2021.
5. Xiaotong Nie, Motoaki Hiraga, and Kazuhiro Ohkura. Visualizing deep Q-learning to understanding behavior of swarm robotic system. In *Proceedings of the 23rd Asia Pacific*

Symposium on Intelligent and Evolutionary Systems, pp. 118–129, 2019.

6. Motoaki Hiraga, Yasumasa Tamura, and Kazuhiro Ohkura. Effects of robot collisions on collective behavior in evolutionary robotic swarms. In *Proceedings of the 3rd International Symposium on Swarm Behavior and Bio-Inspired Robotics*, pp. 303–310, 2019.
7. Motoaki Hiraga, Yufei Wei, and Kazuhiro Ohkura. Evolving collective cognition of robotic swarms in the foraging task with poison. In *Proceedings of the 2019 IEEE Congress on Evolutionary Computation*, pp. 3205–3212, 2019.
8. Daichi Morimoto, Motoaki Hiraga, and Kazuhiro Ohkura. Towards a robotic swarm using deep neuroevolution: An experimental study in path formation. In *Proceedings of the 22nd Asia Pacific Symposium on Intelligent and Evolutionary Systems*, pp. 77–80, 2018.
9. Motoaki Hiraga and Kazuhiro Ohkura. Effects of body size on autonomous specialization and congestion of robotic swarms. In *Proceedings of the 22nd Asia Pacific Symposium on Intelligent and Evolutionary Systems*, pp. 85–92, 2018.
10. Motoaki Hiraga and Kazuhiro Ohkura. Evolutionary emergence of path formation with autonomous specialization in a robotic swarm. In *Proceedings of the 2018 Conference on Artificial Life*, pp. 526–527, 2018.
11. Kazuhiro Ohkura and Motoaki Hiraga. Congestion: A key factor for division of labor in a robotic swarm. In *Proceedings of the First International Conference on Digital Practice for Science, Technology, Education, and Management*, pp. 66–71, 2018.
12. Motoaki Hiraga, Toshiyuki Yasuda, and Kazuhiro Ohkura. Evolutionary acquisition of congestion management of a robotic swarm in a path formation task. In *Proceedings of the 2nd International Symposium on Swarm Behavior and Bio-Inspired Robotics*, pp. 141–146, 2017.
13. Kazuhiro Ohkura, Toshiyuki Yasuda, and Motoaki Hiraga. Observing path formation behavior in evolutionary swarm robotic systems. In *Proceedings of SICE Annual Conference 2016*, pp. 1220–1223, 2016.
14. Toshiyuki Yasuda, Motoaki Hiraga, Akitoshi Adachi, and Kazuhiro Ohkura. Consideration regarding the reduction of reality gap in evolutionary swarm robotics. In *Proceedings of the Tenth International Conference on Swarm Intelligence*, pp. 294–295, 2016.

Domestic Conferences

1. 呉田和優, 潮崎直哉, 森本大智, 平賀元彰, 大倉和博. World Models を適用した Deep Neuroevolurion によるロボティクスワームの群れ行動生成. 日本機械学会中国四国学生会第52回学生員卒業研究発表講演会, 12a3, 2022.
2. 廣川卓海, 平賀元彰, 大倉和博. 漸進進化を用いた MBEANN によるロボティックスワームの制御器設計とその進化過程解析. 第22回計測自動制御学会システムインテグレーション部

門講演会論文集, 1B3-02, pp. 212–217, 2021.

3. 塚本遙日, 森本大智, 平賀元彰, 大倉和博, 棟朝雅晴. 段差環境での多脚自律ロボットスワームによるチェイン生成. 第22回計測自動制御学会システムインテグレーション部門講演会論文集, 1B3-03, pp. 218–223, 2021.
4. 桃崎眞, 森本大智, 平賀元彰, 大倉和博. Deep Neuroevolution によるロボティックスワームの群れ行動生成：障害物の配置が与える影響の一考察. 第22回計測自動制御学会システムインテグレーション部門講演会論文集, 1B3-07, pp. 238–243, 2021.
5. 潮崎直哉, 森本大智, 平賀元彰, 大倉和博. Deep Neuroevolution における進化計算方式の検討：ロボティックスワームの場合. 計測自動制御学会システム・情報部門学術講演会 2021 講演論文集, SS5-1-2, pp. 77–81, 2021.
6. 廣川卓海, 平賀元彰, 大倉和博. 構造進化型人工神経回路網によるロボティックスワームの制御器設計. ロボティクス・メカトロニクス講演会2021講演論文集, 2P1-G06, 2021.
7. 潮崎直哉, 森本大智, 平賀元彰, 大倉和博. LSTM を用いた Deep Neuroevolution によるロボティックスワームの群れ行動生成. ロボティクス・メカトロニクス講演会2021講演論文集, 2P1-G04, 2021.
8. 塚本遙日, 森本大智, 平賀元彰, 大倉和博, 棟朝雅晴. Neuroevolution による多脚自律ロボットスワームの群れ行動生成：二点間往復タスクの場合. 第65回システム制御情報学会研究発表講演会講演論文集, Gse-05-3, pp. 851–858, 2021.
9. 渡辺優, 平賀元彰, 大倉和博. 二重倒立振子制御問題を用いた NEAT と MBEANN の比較. 第65回システム制御情報学会研究発表講演会講演論文集, Gse-01-2, pp. 107–113, 2021.
10. 潮崎直哉, 森本大智, 平賀元彰, 大倉和博. Deep Neuroevolution を適用した LSTM によるロボティックスワームの群れ行動生成. 日本機械学会中国四国学生会第51回学生員卒業研究発表講演会, 13b2, 2021.
11. 塚本遙日, 森本大智, 平賀元彰, 大倉和博. 不整地環境下での多脚自律ロボットスワームの群れ行動生成. 日本機械学会中国四国学生会第51回学生員卒業研究発表講演会, 13b1, 2021.
12. 渡辺優, 平賀元彰, 大倉和博. 構造進化型人工神経回路網の設計方針に対する一考察. 日本機械学会中国四国学生会第51回学生員卒業研究発表講演会, 13a5, 2021.
13. 廣川卓海, 平賀元彰, 大倉和博. NEAT と MBEANN によるロボティックスワームの制御器設計. 日本機械学会中国四国学生会第51回学生員卒業研究発表講演会, 13a4, 2021.
14. 吉崎豪, 福頼征弥, 平賀元彰, 大倉和博. PredNet による群れモデルの長期的行動予測. 日本機械学会中国四国学生会第51回学生員卒業研究発表講演会, 05b3, 2021.
15. 桃崎眞, 森本大智, 平賀元彰, 大倉和博. Deep Neuroevolution によるロボティックスワームの群れ行動生成とその制御器解析. 第21回計測自動制御学会システムインテグレーション部門講演会論文集, 3B2-10, pp. 2366–2370, 2020.
16. 福頼征弥, 平賀元彰, 大倉和博, 松村嘉之. PredNet を用いた群れモデルの行動予測. 第21回計測自動制御学会システムインテグレーション部門講演会論文集, 3B2-09, pp. 2362–2365, 2020.
17. 森本大智, 平賀元彰, 大倉和博, 松村嘉之, 棟朝雅晴. Deep Neuroevolution による多脚自

律ロボットスワームの群れ行動生成. 2020年度人工知能学会全国大会（第34回）論文集, 2M5-OS-3b-02, 2020.

18. 内田隼, 森本大智, 平賀元彰, 大倉和博. Neuroevolution によるロボティックスワームの合意形成. ロボティクス・メカトロニクス講演会2020講演論文集, 1P1-I05, 2020.
19. 福頼征弥, 平賀元彰, 大倉和博, 松村嘉之. 対話型進化計算を用いたスワームシステムの群れ行動設計. ロボティクス・メカトロニクス講演会2020講演論文集, 1P1-I01, 2020.
20. 桃崎眞, 森本大智, 平賀元彰, 大倉和博. ロボティックスワームの群れ行動生成における Deep Neuroevolution の拡張に関する一考察. 第64回システム制御情報学会研究発表講演会講演論文集, GS17-5, pp. 786–792, 2020.
21. 桃崎眞, 森本大智, 平賀元彰, 大倉和博. Deep Neuroevolution によるロボティックスワームのパティオ環境での群れ行動生成. 日本機械学会中国四国支部第58期総会・講演会講演論文集, 12c5, 2020.
22. 森本大智, 平賀元彰, 大倉和博. パティオ環境におけるロボティックスワームの群れ行動生成：DeepNeuroevolution に基づくアプローチ. 第20回計測自動制御学会システムインテグレーション部門講演会論文集, 3D1-12, pp. 2746–2751, 2019.
23. Yuxi Lu, Daichi Morimoto, Motoaki Hiraga, and Kazuhiro Ohkura. Deep neuroevolution in collective transport with swarm robotics system. 第28回計測自動制御学会中国支部学術講演会論文集, 3B-6, pp. 71–72, 2019.
24. Wenqian Yu, Ruipeng Ji, Motoaki Hiraga, and Kazuhiro Ohkura. Evolving collective cognition behavior of robotic swarms in the sorting and foraging food. 第28回計測自動制御学会中国支部学術講演会論文集, 1B-3, pp. 13–14, 2019.
25. Ruipeng Ji, Wenqian Yu, Motoaki Hiraga, and Kazuhiro Ohkura. Effects of food radius on evolving swarm robotics system in the collective food foraging task. 第28回計測自動制御学会中国支部学術講演会論文集, 1B-2, pp. 11–12, 2019.
26. 鉄山法隆, 森本大智, 平賀元彰, 大倉和博, 松村嘉之. 深層強化学習によるロボティックスワームの群れ行動生成とその制御器解析. ロボティクス・メカトロニクス講演会2019講演論文集, 1P2-H10, 2019.
27. 森本大智, 平賀元彰, 大倉和博, 松村嘉之. Deep Neuroevolution によるロボティックスワームの二点間往復タスクにおける群れ行動の生成. 2019年度人工知能学会全国大会（第33回）論文集, 3D3-OS-4a-04, 2019.
28. 福頼征弥, 平賀元彰, 大倉和博, 松村嘉之. Predator-Prey モデルにおける群れ行動生成とその定量的計測手法. 第63回システム制御情報学会研究発表講演会講演論文集, GSe01-7, pp. 1039–1044, 2019.
29. 森本大智, 平賀元彰, 大倉和博, 松村嘉之. DQN と Deep-Neuroevolution によるロボティックスワームの群れ行動生成と解析. 第63回システム制御情報学会研究発表講演会講演論文集, GSe04-5, pp. 1228–1234, 2019.
30. 森本大智, 平賀元彰, 大倉和博. Deep-Neuroevolution に基づくロボティックスワームの群れ行動生成. 第19回計測自動制御学会システムインテグレーション部門講演会論文集, 1C6-10,

pp. 876–881, 2018.

31. 平賀元彰, 森本大智, 福頼征弥, 大倉和博. 進化型スワームロボティクスにおける超冗長性と混雑がもたらす創発的機能分化. 2018年度精密工学会北海道支部学術講演会講演論文集, A-14, pp. 27–28, 2018.
32. 保田俊行, 平賀元彰, 大倉和博. 進化型スワームロボットシステムの拡張性に個体設計が及ぼす影響. 第17回計測自動制御学会システムインテグレーション部門講演会講演論文集, 1D3-6, pp. 298–300, 2016.
33. 平賀元彰, 保田俊行, 大倉和博. 進化型スワームロボティクスにおける個体設計とシステム規模の関係. 2016 IEEE SMC Hiroshima Chapter 若手研究会講演論文集, pp. 135–137, 2016.
34. 平賀元彰, 保田俊行, 大倉和博. 進化型ロボティックスワームにおけるリアリティギャップの縮小に関する一考察. ロボティクス・メカトロニクス講演会2016講演論文集, 1A1-05a3, 2016.