Automatic Design of Controllers for a Multi-Legged Robotic Swarm

(多脚ロボティックスワームのための制御器の自動的設計)

Daichi Morimoto
(森本 大智)

Mechanical Engineering Program
Graduate School of Advanced Science and Engineering
Hiroshima University

September 2023

Abstract

This thesis presents automatic designs of controllers for a multi-legged robotic swarm. Swarm robotics aims to generate desirable collective behaviors based on local interactions of numerous autonomous robots. Many studies in swarm robotics have discussed how to design robot controllers for defining the interactions of robots. However, in addition to the controller settings, the local interactions between robots or robots and the environment also depend on robot specifications, such as body structure, movability, and sensor settings. Based on this point of view, we can find that studies in swarm robotics typically employed mobile robots driven by wheels or vibrating rigid legs. In general, by using these types of robots, collective behaviors are limited in two-dimensional space. To overcome this limitation, several studies utilize different types of robots, such as unmanned aerial vehicles, underwater robots, and multi-legged robots.

This thesis focuses on generating collective behaviors of a multi-legged robotic swarm. Multi-legged robots are expected to operate in rough terrains that are difficult for wheeled-mobile robots to move well. Additionally, multi-legged robotic swarms are expected to exhibit novel collective behaviors inspired by the self-assembly of army ants. However, designing a robot controller becomes a challenging problem because a controller decides not only how to coordinate a large number of robots based on local observations, but also how to coordinate a large number of actuators in individual robots. Therefore, multi-legged robotic swarms raise a new problem domain as a combination of two types of large-degree-of-freedom controls. To address this problem, this thesis employs automatic design methods used in swarm robotics. This thesis contributes to the swarm robotics community by following two aspects.

First, this thesis presents evolutionary robotics approaches for designing controllers of a multi-legged robotic swarm. Evolutionary robotics is a technique to utilize evolutionary algorithms for designing robot controllers. In swarm robotics, evolutionary robotics have shown promising results for designing collective behaviors. In addition, it has succeeded in designing gaits for multi-legged robots. This thesis aims at the hybridization between evolving gaits and collective behaviors. Experimental results showed that the evolutionary robotics approach successfully designed controllers of a multi-legged robotic swarm in several task scenarios, such as in rough terrains or generating a three-dimensional collective behavior.

Second, this thesis presents the reinforcement learning-based approach. Reinforcement learning is the common machine learning method for obtaining an agent's behaviors through interactions with the environment. Generally, designing a controller for robotic swarms becomes a challenging problem for reinforcement learning due to the dynamic environmental settings or local observations. On the other hand, recent trends in reinforcement learning have achieved remarkable results by employing deep neural networks as the function approximator for an agent policy or value functions. This thesis shows that a deep reinforcement learning algorithm successfully designed a controller of a multi-legged robotic swarm.

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. Soichi Ibaraki, associate Prof. Yu Kawano, and Prof. Yoshiyuki Matsumura, for revising the thesis and providing insightful comments.

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 JP21J23095.

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.

July 2023
Daichi Morimoto

Contents

List of Figures

List of Tables

Chapter 1

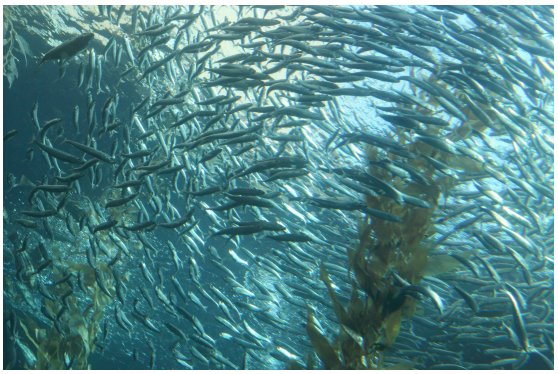
Introduction

Natural organisms have provided not only large interests in science but also great inspirations for engineering. For example, in robotics, a variety of bio-inspired robots have been developed in the past half-century, such as insects [15, 17, 91], quadruped mammals [13, 42, 125], birds [33, 120], fish [9, 70], and snakes [68, 159, 162]. In the machine learning field, the mathematical models for biological nerve systems (i.e., artificial neural networks) have shown great success and raise recent artificial intelligence trends [46, 83]. Add to these examples, *bio-inspired engineering* has shown a wide range of achievements such as in materials, textiles, and chemistry [11, 126]. Nature and living things will give fruitful ideas for studies of engineering.

One of the interesting phenomena in natural creatures is the ability to show collective behaviors by forming a group of individuals [136, 143, 155]. Fig. 1.1 shows examples of collective behaviors exhibited by natural swarm systems. In flocks of birds or schools of fish, aggregation and coordinated motions are hypothesized to reduce the risk of attack by predators. Ant shows cooperative transport behavior for heavy objects that are hard for a single individual to bring back to the nest. Honey bee shows *waggle dance* to share useful information for mates, such as direction and distance to flowers, or location of new nest candidate. These phenomena are remarkable in that there is no central control mechanism to supervise the system, and they emerged from the autonomous and decentralized actions of individuals. A variety of collective behaviors and their honed mechanisms gathered large interests from scientists, and raise a new research domain, *swarm intelligence* [12, 71] which is a subfield of artificial intelligence. The studies of swarm intelligence include a wide of examples, such as exploring the mechanism of collective behaviors [31, 82, 113], developing numerical optimization algorithms [26, 69, 73], and applications for multi-robot systems [7, 8].

Swarm robotics is a promising approach to applying swarm intelligence to a large number of physically-embodied autonomous robots [14, 27, 51, 123]. In [123], swarm robotics is defined as *the study of how large number of relatively simple physically embodied agents*

(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 Martin LaBar, licensed under CC BY-NC 2.0).

Fig. 1.1. Examples of collective behavior exhibited by natural swarm systems.

can be designed such that a desired collective behavior emerges from the local interactions among agents and between the agents and the environment. Similar to biological swarms, the robot system in swarm robotics (i.e., *robotic swarm*) is expected to show capability beyond the single individual. In addition, the robotic swarm is also expected to show system-level properties, such as (i) fault torrence: when some of the robots become failures, the system keeps to operate in the task with degradation of performance, (ii) flexibility: the system can work in different task scenarios from the design time, (iii) scalability: the system cope with the changes about the number of robots. To realize these properties, maturing studies have tried to emulate several collective behaviors, such as aggregation [29, 44], pattern formation [124, 132], path formation [111, 137], collective transport [47, 48], and collective decision making [141, 151].

Designing interactions between robots or robots and environments is an essential part to generate collective behaviors of robotic swarms. Many studies in swarm robotics have discussed how to design a robot controller for defining the local interactions of robots. On the

other side, robot specifications such as body structure, actuator settings, and sensor settings are determined before or in parallel with the controller design. Obviously, interactions between robots or robots and the environment depend on not only the controller setting but also robot specifications. Based on this point of view, we can find that many studies in swarm robotics have employed mobile robots driven by wheels or vibrating rigid legs, which move on relatively flat surfaces [29, 47, 75, 96, 111, 122, 137, 141]. By using these types of robots, researchers are able to focus on designing collective behavior because it is relatively easy to modularize the primitive motions of each robot, such as moving forward or backward, turning right or left, and stopping. On the other hand, collective behaviors generated by a robotic swarm are usually limited in two-dimensional space. To overcome this limitation, some studies in swarm robotics or multi-agent systems employed other types of robots, such as underwater robots [9, 163], unmanned aerial vehicles [95, 154], and multi-legged robots [114]. New types of robots will make it more difficult to control a single robot, while they allow a robotic swarm to operate in a wider range of environments or tasks.

This thesis focuses on generating collective behaviors of a multi-legged robotic swarm. The multi-legged robots show three-dimensional motions, such as climbing steps or obstacles by coordinating a large number of joints in each leg. Therefore, the multi-legged robotic swarm is expected to operate in rough terrain fields where wheeled mobile robots can not move well. Additionally, the multi-legged robotic swarm is also expected to show novel collective behaviors inspired by legged creatures. For example, army ants are known for *self-assembly*, which is constructing huge structures such as walls, bridges, rafts, and bivouacs by using their bodies [1]. By emulating this behavior, the multi-legged robotic swarm is expected to show cooperative behavior in overcoming the terrain or obstacles that are difficult for a single robot to traverse. These behaviors seem to be effective in various kinds of missions, such as exploring or rescuing in unknown environments.

On the other hand, compared with wheeled-mobile robots, designing a controller for a multi-legged robotic swarm becomes a challenging problem because the designer has to consider not only how to coordinate a large number of robots based on local observations, but also how to coordinate a large number of actuators in individual robots. Therefore, a multi-legged robotic swarm raises a new problem domain as a combination of two types of large-degree-of-freedom controls. As far as we know, only a few studies address this problem [114]. Additionally, this issue seems to be hard and time-consuming for the common approach in swarm robotics, which is *behavior-based design methods* [14]. In this method, the trial and error process is conducted for designing a robot controller, where the designer assumes the behaviors of each robot and updates them until the desired collective behavior is obtained. In the multi-legged robotic swarm, the behavior-based design becomes an extremely hard problem for the following reasons.

- It is too difficult to assume the modularized motions of each robot that are required for the collective behavior; there are a huge number of robot states because interactions among robots or robots and environments occur in three-dimensional space. If designing self-assembling-like behavior, designers should consider overlapping among robots.
- Even if the required motions of each robot are already known, it is also hard to realize the motion by coordinating a large number of joints in individual robots. When climbing obstacles, how to move joints depends on the shape or height of the obstacles. When climbing other robots, robots have to take into account the postures of robots that act as stepping stones.

Therefore, this thesis focuses on another approach for designing controllers, *automatic design methods* [14].

Automatic design methods are approaches to designing robot controllers automatically by using several computation techniques. In this method, controller design problems are typically cast into optimization problems. Therefore, designers only need to determine the final goal or subgoals of the task; designers can reduce efforts in developing controllers. The most used approach in automatic design methods is evolutionary robotics [35, 108] which utilizes evolutionary computations to optimize the parameters of a robot controller. Evolutionary robotics has achieved several collective behaviors [5, 29, 39, 47, 55, 137]. Another approach is reinforcement learning which is a class of machine learning problems; an agent learns a behavior during trial-and-error interactions with the environment. Reinforcement learning is also applied to robotic swarms [60, 161], while it is a relatively minor choice due to the difficulties of local observations or dynamic environmental properties. In addition, both techniques have been utilized to design gaits for multi-legged robots [20, 49, 53, 107, 119, 148, 152]. With the rise of artificial intelligence fever, many researchers are interested in these techniques for designing robot locomotions. However, it is unclear whether these machine learning approaches are effective in the combination problem of designing a gait and collective behavior. The main purpose of this thesis is to show that the automatic design method is a promising way to design controllers of a multi-legged robotic swarm.

1.1 Aim and Objectives

This thesis presents automatic designs of controllers for a multi-legged robotic swarm. In the swarm robotics community, few studies worked on a combination of a multi-legged robot and a robotic swarm. This thesis contributes to the swarm robotic community to show the effectiveness of the automatic design methods with the following aspects.

First, this thesis employs an evolutionary robotics approach to generate collective behaviors of a multi-legged robotic swarm. Compared with the wheeled-mobile robots, multi-legged robots required a complicated control scheme for basic movements. The evolutionary robotics

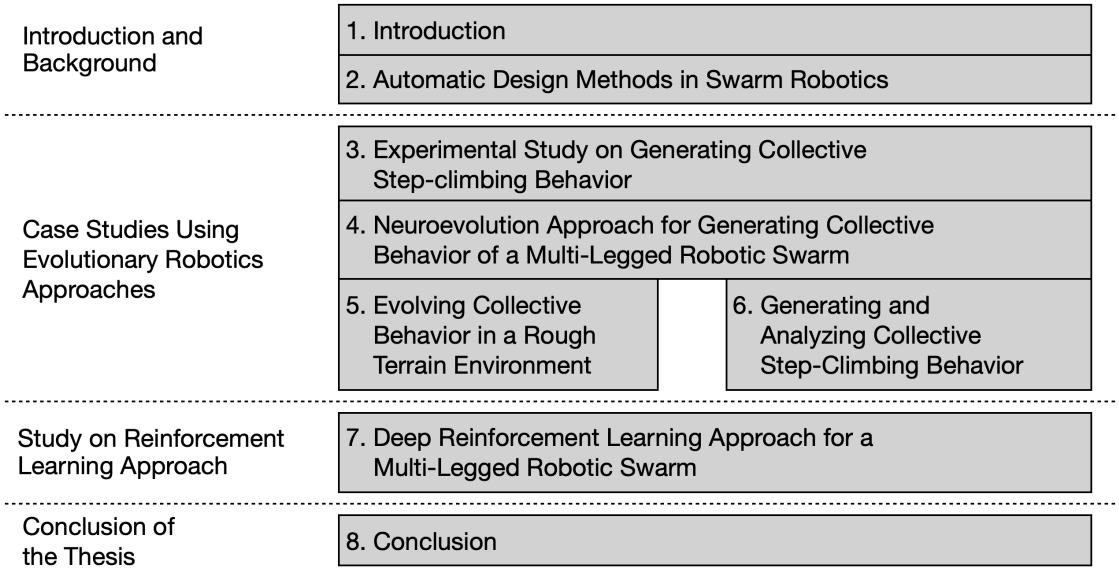


Fig. 1.2. The structure of the thesis..

approach is succeeded in designing both gait of a single robot and collective behavior of a robotic swarm. Therefore, hybridization between these results in evolutionary robotics become a promising way. By applying the approach, this thesis aims to generate collective behaviors in not only flat but also rough terrains. In addition, it aims to generate a three-dimensional collective behavior inspired by the self-assembly of army ants. These novel collective behaviors are expected to give insights into how to use robotic swarms in real-world applications.

Second, this thesis examines the applicability of reinforcement learning to design a controller of a multi-legged robotic swarm. So far, the swarm robotics problem seems to be hard to solve with the reinforcement learning approach because the learning processes are typically taken place at an individual level, while collective-level behaviors are finally required. On the other hand, reinforcement learning has an advantage in computational cost compared with the evolutionary robotics approach. Additionally, with the recent trend in artificial intelligence, several novel and sophisticated reinforcement learning algorithms are proposed [40, 50, 99, 129]. This thesis employs a proximal policy optimization algorithm [129] which is one of the most popular reinforcement algorithms. This approach clarifies the potential of reinforcement learning to design a controller of a multi-legged robotic swarm.

1.2 Structure of the Thesis

As described above, this thesis presents automatic design approaches for designing controllers of a multi-legged robotic swarm. Fig. 1.2 illustrates an overview of the thesis structure. This thesis is roughly composed of four parts. The first part (Chapters 1 and 2) describes

the introduction of the thesis and computation techniques used in the thesis, respectively. The second part (from Chapter 3 to 6) shows case studies based on evolutionary robotics approaches for generating collective behaviors of a multi-legged robotic swarm. In addition, the third part (Chapter 7) presents the reinforcement learning-based approach which is an alternative to the evolutionary robotics approach. At last, the fourth part (Chapter 8) concludes this thesis. The content of each chapter is briefly summarized as follows.

- Chapter 2 describes automatic design methods in swarm robotics. The first section provides introductions to evolutionary robotics with some related topics, such as evolutionary computations, neuroevolution, and related work. Second, this chapter describes the overview of reinforcement learning with basic topics and recent trends.
- Chapter 3 shows the first example of the evolutionary robotics approach for the three-dimensional collective behavior of a multi-legged robotic swarm. As mentioned above, a combination of designing gait and collective behavior is a highly complex problem. This chapter employs both the manual design and evolutionary design as an experimental approach. In the studies of multi-legged robots, the central pattern generator (CPG) which is inspired by spine creatures is a primary option for the robot controller. Based on this background, the controller of a robotic swarm is developed by connecting an artificial neural network (ANN) to the hand-tuned CPG. The synaptic weights of ANN are optimized by evolutionary computation to that the output of ANN affects the CPG signal and generates a collective behavior. The experimental results showed that the proposed method successfully designed three-dimensional collective behavior. Additionally, the analysis for the controller indicated that the independent evolution trials show similar tendencies regarding obtained controller properties.
- Chapter 4 summarized the pure neuroevolution approach. The proposed method in Chapter 3 could skip the evolution of basic gait. However, the tuning of CPG highly depends on the robot's specifications. Therefore, re-tuning takes a lot of time when robot settings are updated. This chapter proposes the approach in which the basic gait of single robots is also generated by evolutionary design: the pure neuroevolution approach. In general, a multi-legged robot is viable for huge aspects of gaits due to the large degrees of freedom, and sometimes shows unnatural (not similar to natural creatures) gait which is may unsuitable to achieve the task by mimicking natural creatures. In this chapter, the neuroevolution approach shows that robot gait which has similar features to legged animals is successfully obtained by incorporating intuition-based objective functions into the fitness evaluations. In addition, the neuroevolution approach successfully designed a control scheme for collective behavior, i.e., how to coordinate a large number of joints on legs based on local observations to show the coordinated motion of robots.

- Chapter 5 shows the collective behavior design in a rough terrain environment. The multi-legged robotic swarm is expected to operate in fields with rugged surfaces or uneven surfaces. Based on the result of Chapter 4, this chapter aims to achieve the path formation task in not only flat but also rough terrains. In the experiment, the rough terrain is developed by cuboid blocks that are arranged like a lattice. Therefore, the terrain aspect is parametrized by the height of each block; the swarm performance is quantitatively discussed with the terrain settings. The experimental results showed the evolutionary robotics approach successfully achieved a path formation task in rough terrain environments. The results also showed that incremental evolution is an effective way to design a controller in a rough terrain setting.
- Chapter 6 focuses on generating and analyzing the collective step-climbing behavior. The proposed approach in Chapter 4 is applied to generate a three-dimensional collective behavior. In this task, robots aim to climb the step which is too high for a single robot; robots have to utilize other robots as stepping stones. Therefore, the robot controller is required to generate two types of actions: climbing the step or other robots and acting as a stepping stone. This chapter shows that the neuroevolution approach is a promising way to design a robot controller for the collective step-climbing task. Additionally, evolved controllers are analyzed to clarify what kind of behaviors are obtained. Based on the preliminary experiments, robots seem to show not only climbing of other robots or steps but also developing helpful stepping stones for achieving the task. In this chapter, measurement factors are designed to analyze robot behaviors. The experimental results showed that the transitions of measurement factors support the hypothesis about obtained behaviors.
- Chapter 7 discusses the reinforcement learning-based approach. In former Chapters, evolutionary robotics approaches showed successful results on a multi-legged robotic swarm, similar to traditional automatic design approaches in swarm robotics. On the other hand, evolutionary computations are often denoted disadvantages with high computational costs, such as massive resources for parallel evaluations or long-term calculation due to the generation-based update. The automatic design methods of swarm robotics also involve reinforcement learning which is a relatively minor approach; the robotic swarm scenario seems a hard problem for reinforcement learning. However, the computational cost required for RL is relatively lower than evolutionary robotics. Moreover, recent trends in artificial intelligence grow novel deep reinforcement learning (DRL) algorithms. This chapter examines the applicability of DRL for a multi-legged robotic swarm. Proximal policy optimization (PPO) which is one of the most popular DRL algorithms is applied to the task of forming a line. The result showed that the PPO successfully designed the controller of a multi-legged robotic swarm.
- Chapter 8 concludes the thesis and discusses future research directions.

Chapter 2

Automatic Design Methods in Swarm Robotics

This chapter describes the automatic design methods, which are the main approaches to designing robot controllers in this thesis. In swarm robotics, many researchers have discussed the methodology of designing controllers of a robotics swarm to realize desirable collective behaviors from local interactions among robots. The controller design methods are classified into two categories: behavior-based design methods and automatic design methods [14]. The behavior-based design method follows a trial-and-error process; the controller consisting of the primitive actions of robots is improved by human designers until the desired collective behavior is obtained. Generally, this approach is easily accomplished if the mathematical model for the collective behavior is available, otherwise, it requires a human designer's intuitions or experiences. In contrast, the automatic design method employs computational techniques to reduce efforts by the designer, while it sometimes suffers from computational costs or the reality gap [63, 134]. This thesis employs automatic design methods for a multi-legged robotic swarm problem because it seems a promising way to simultaneously design a gait and collective behavior. The automatic design method is divided into two main domains: evolutionary robotics and reinforcement learning. The remaining parts of this chapter briefly describe both methods.

Section 2.1 introduces evolutionary robotics with related topics, such as evolutionary computation, neuroevolution, and achievements in the method. Afterward, Section 2.2 briefly describes reinforcement learning with basic ideas of the method and recent trends.

2.1 Evolutionary Robotics

Evolutionary robotics (ER) is defined as the research field or the approach to designing robot properties, such as control software, body morphology, and sensor-motor properties

by using evolutionary computation techniques [109, 149]. The rise of ER started at the University of Sussex [19, 52], the Swiss Federal Institute of Technology in Lausanne [36], and the University of Southern California [85] in the early 1990s. These studies gathered a lot of attention from worldwide and initiated the research field. Up to the present, ER approaches have been applied in a variety of domains, such as evolving gaits of multi-legged robots [20, 59, 119, 152], designing collective behaviors of robotic swarms [29, 47, 137], and morphogenetic evolution of robots [21, 76, 116]. In the rest of this section, the related topics to the ER are described as follows; evolutionary computation in Section 2.1.1, neuroevolution in Section 2.1.2, and applications of ER in Section 2.1.3.

2.1.1 Evolutionary Computation

Evolutionary computation is one of the subfields in computer science, which develops computational techniques inspired by natural evolution. The main inspiration comes from the Darwinian principle [23], i.e., natural selection or survival of the fittest; the differences among individuals in a population of creatures are *selected* by nature based on whether adapt to the environment. In general, the computational techniques inspired by the natural selection mechanism are called *evolutionary algorithms*. The initial studies of evolutionary algorithms appeared in the 1960s, such as evolutionary programming (EP) [37, 38], genetic algorithm (GA) [45, 56], and evolutionary strategies (ES) [10, 130]. In the early 1990s, these methods are categorized as part of evolutionary computations, and genetic programming (GP) [77, 78] was proposed as the new method. At present, these four algorithms are fundamental parts of the evolutionary algorithm. On the other side, several population-based optimization algorithms have been proposed, such as differential evolution (DE) [117, 140], particle swarm optimization (PSO) [72, 115], ant colony optimization (ACO) [24, 25], bat algorithm [160], and grey wolf optimizer [98]. Not all these algorithms follow natural evolution mechanisms (PSO, ACO, and ABC algorithms are categorized as *swarm intelligence* [16, 71]), although they have a similar framework and processes to evolutionary algorithms. These techniques have a lot of attention from researchers and form a large research community.

Algorithm 1 shows the typical process conducted by evolutionary algorithms. Generally, the solution candidates for the target problem are called *individual* or *phenotype* inspired by the biological terminology. The phenotype is constructed by *decoding* the *genotype* which is a parameter vector for deciding phenotype properties (contrary, the mapping from phenotype to genotype is called *encoding*). The evolutionary algorithms started with the initialization of individuals. In the *Evaluation* phase, the performance of each individual is calculated by the fitness function (i.e., objective function). Afterward, the algorithm starts a generational loop. At first, a part of individuals are selected as *parents* based on their fitness values. In the genetic operation phase, several operators (e.g., recombination or mutation) are

Algorithm 1 Pseudo-code of a typical evolutionary algorithm.

1: *Initialization*: Generate an initial population of individuals;
2: *Evaluation*: Calculate the fitness value of each individual;
3: **repeat**
4: *Selection*: Select parents from the current population;
5: *Recombination*: Generate offspring by recombining parents;
6: *Mutation*: Mutate each offspring;
7: *Evaluation*: Calculate the fitness value of each offspring;
8: *Selection*: Select survivors for the next generation;
9: **until** Terminal condition;

applied to the genotypes of parents for generating offspring. Each offspring is also evaluated for its performance by the fitness function. Finally, the *survivor selection* phase selects the individuals among parents and offspring, which become parent candidates in the next generation. These processes continue until the terminational conditions (e.g., desirable performances or maximum loop number) are satisfied. In the algorithm, the genetic operator corresponds to enlarge the exploring area in parameter space, whereas parent or survivor selections are responsible for focusing the population in a promising area. By repeating processes, the population probabilistically finds better solutions for the target problem. These basic mechanisms of evolutionary algorithms are highly generalizable because the algorithm only needs to obtain the fitness values.

2.1.2 Neuroevolution

Neuroevolution is one of the applications of evolutionary computation, which aims to optimize artificial neural networks [34]. This thesis employs neuroevolution as a main approach to designing a controller of a multi-legged robotic swarm. The rest of this section describes related topics to neuroevolution: an artificial neural network and details of the approach.

Artificial neural networks (ANNs) are known as mathematical models for the nerve systems of natural creatures [46, 62]. In the ANNs, the artificial neurons are designed to emulate the properties of neural cells. The important properties of neural cells are modeled in ANNs as follows:

- The output of neural cells affects the membrane voltage of connected neurons.
- The synaptic connections are categorized into positive and negative connections; the positive synapse increases the membrane voltage of connected neurons, and vice versa.
- Each neuron outputs a pulse signal if the voltage is higher than the threshold. The threshold-based response of neurons acts as a non-linear function from input to output.

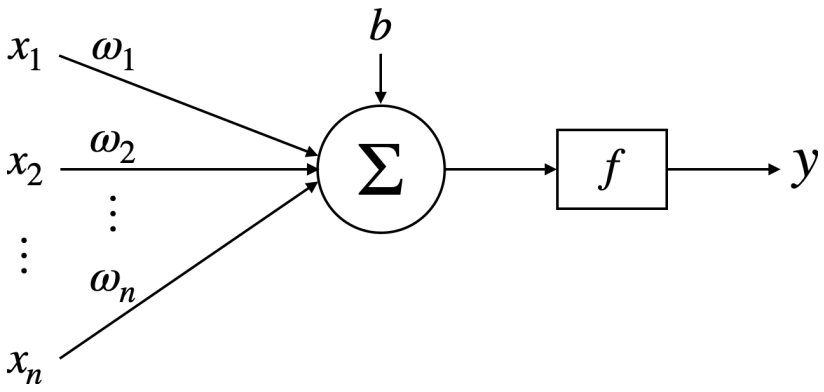


Fig. 2.1. Artificial neuron model.

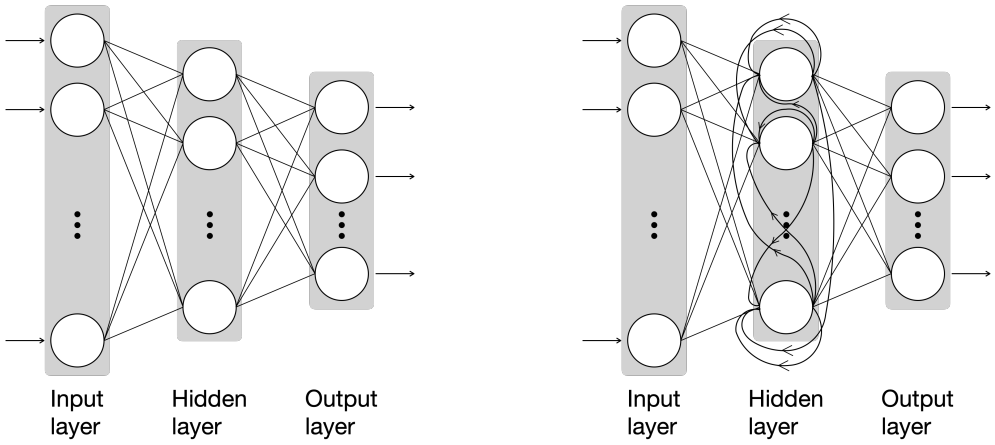
- The intensity or type of synaptic connections is updated according to the frequency of receiving pulses; this property is called synaptic plasticity. The plasticity enables the network to memorize information.

According to these properties, McCulloch and Pitts proposed a formal neuron model [94] in 1943. Based on the formal model, the modern ANNs employ an artificial neuron model, which is illustrated in Fig. 2.1. In this model, the output of neuron y is calculated by the following equation:

$$y = f\left(\sum_{i=1}^{n} w_i x_i + b\right), \tag{2.1}$$

where $x_1, \ldots, x_n$ are input signals from other neurons, w_i is the synaptic weight for x_i, and b is the bias. The function f is called the activation function that emulates a non-linearity in neuron cells. Usually, the functions such as sigmoid, hyperbolic tangent, and rectified linear units are employed as the activation function. The ANNs consist of numerous artificial neurons as the directional network. Fig. 2.2 shows the commonly used ANNs, which have layered structures with input, hidden, and output layers. The standard structure is the feedforward neural network (Fig. 2.2(a)), in which the computational process for output is conducted in a single direction (from input to output layer). In contrast, the recurrent neural network (Fig. 2.2(b)) has recursive connections among neurons, which take into account the past neuron's states. Therefore, the recurrent network is employed for tasks with sequential data, such as text classification or speech recognition.

Due to the synaptic plasticity, the ANNs show learnability; the output value corresponding to the input can be changed by arranging the synaptic weights. To obtain the desired output of ANNs, the network parameters such as synaptic weights or biases have to be updated; parameter updates are called *training*. The most common method for training ANNs is stochastic gradient descent (SGD) with backpropagation (BP) [87, 93]. The SGD is a powerful method for rapid training of ANNs, while it sometimes suffers from



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

Fig. 2.2. Standard structures of artificial neural networks.

gradient vanishing problems, local optimums, and computational complexity for recurrent connections [46, 83]. As another approach, neuroevolution utilizes evolutionary algorithms (i.e., population-based optimization process) to train ANNs. In this approach, the genotype typically consists of synaptic weights and biases of ANNs. The fitness is given by the performance of ANNs on the target problems. Typically, neuroevolution approaches are conducted with fixed network topologies. On the other hand, advanced approaches are proposed to evolve not only synaptic weights but also network structure; the human designer can reduce efforts on exploring network topology for the task [138].

2.1.3 Evolutionary Robotics Approach for Designing Controllers

In this section, applications of the evolutionary robotics approach are described with the topics of sections 2.1.1 and 2.1.2. Evolutionary robotics (ER) is a technique to design robot controllers or even robot embodiment by using evolutionary algorithms. The evolutionary computation described in section 2.1.1 is a fundamental technique to execute an ER approach. The neuroevolution described in section 2.1.2 is a common option of ER; many studies employ ANNs as robot controllers. According to algorithm 1 in section 2.1.1, the genotype consists of parameters that determine the robot controller or morphological aspects of the robot. The decoded phenotype (i.e., robot controller or the robot itself) is evaluated in the task; the fitness corresponds to the performance of the robot. The rest of this section shows the some of achievements that are related to this thesis: evolving gait and collective behavior.

The evolutionary robotic approach has been employed for designing the gait of multi-legged robots [20, 59, 119, 148, 152]. Hornby et al. [59] designed a gait of the AIBO robotic dog. The robot controller is represented by locomotion module software composed of mathematical functions. A total of 25 parameters are optimized. Valsalam et al. [152] evolved a modular

neural network controller for the gait of the quadruped robot. Symmetry and modularity are introduced into the network to cope with the reality gap. Clune et al. [20] employed the Hyper-NEAT algorithm to evolve the gait of a quadruped robot. This work is more challenging because they evolved not only synaptic weights but also the topology of the network. In all of these works, the main topic is how to coordinate a large number of legs and joints to generate a gait of a robot. Therefore, the evolutionary processes were conducted with a single robot scenario.

The evolutionary robotics approach also succeeded in designing collective behaviors of robotic swarms [5, 29, 47, 137]. In [29], Dorigo et al. evolved an aggregation behavior. The robot controller was represented by a simple neural network that has only input and output layers. Baldassarre et al. [5] generated a flocking behavior of a robotic swarm. The controller also consists of only input and output layers. In [47], Groß et al. generated a collective transport behavior with the recurrent neural network. The controller was evolved by the algorithm based on $(\mu + \lambda)$ evolution strategies. Sperati et al. [137] evolved a robotic swarm in the path formation task. The controller was represented by the feedforward neural network that has a single hidden layer and direct connections from the input to the output layer. In swarm robotics, the most of evolutionary robotics approaches have been conducted with wheeled-mobile robots. In this thesis, the unit robot of the collective systems consists of a large number of legs and joints (i.e., a large degree of freedom). This thesis presents a hybrid approach between designing the gait of a multi-legged robot and designing collective behavior of a robotic swarm in evolutionary robotics.

2.2 Reinforcement Learning

This section briefly describes reinforcement learning (RL). In general, RL is categorized as one of the machine learning methods [66, 145]. In addition, the RL is utilized as the automatic design method in the swarm robotics field (although evolutionary robotics is a major approach). This thesis also employs RL for the controller design of a multi-legged robotic swarm. The rest of this section describes RL as follows: basic topics of RL in section 2.2.1, and recent trends of deep reinforcement learning in section 2.2.2.

2.2.1 Basis for Reinforcement Learning

This section describes the basic idea, terminologies, and framework of reinforcement learning (RL). RL is categorized as one of the machine learning methods; the third party to supervised learning and unsupervised learning. In contrast to supervised learning, the learning subjective (the learner) does not perceive which option is the correct answer for the input data. Instead of that, the learner gets the scalar value which indicates how the selected option is good; the scalar value is usually called a *reward*. Typically, the framework

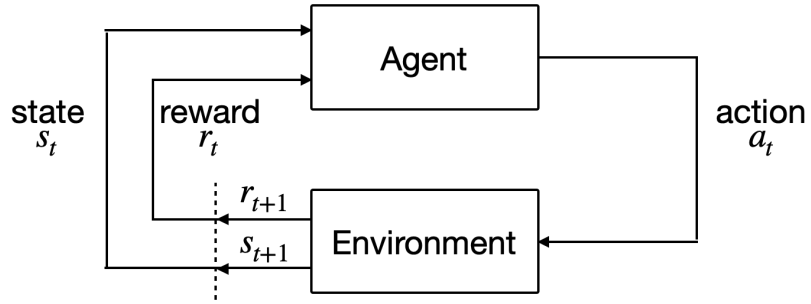


Fig. 2.3. Basic framework for reinforcement learning problems.

of RL is illustrated as Fig. 2.3. Most RL problems are modeled with two fundamental factors: *agent* and *environment*. The agent is defined as the subjective that observes the state of the environment, decides the action, and interacts with the environment. The environment transits its state based on the agent's action and returns the reward to the agent. Generally, the learning process of RL is conducted as follows:

- The agent observes the environment state s_t at timestep t, and decides action a_t.
- The environment transits its state s_t to s_{t+1} based on the agent's action and returns reward r_t to the agent.
- Update timestep ($t \leftarrow t + 1$) and go back to the first step.

The objective of RL is to train the agent for maximizing cumulative reward. To achieve the objective, the agent is required to *explore* what behavior sequence obtains a maximum reward. In addition, the agent has to take into account that an action selected at a timestep will affect consequent rewards. These two features (trial-and-error exploration and delayed reward) are important to characterize RL in the machine learning field.

In [145], Sutton and Barto noted the principle components of RL: the Marcov-decision process, policy, reward signal, value function, and model of the environment. The rest of this section briefly describes these key components. In addition, several categorizations for RL are introduced.

Markov decision process

Markov decision process (MDP) is the stochastic process that presents the state transition by the selected actions [6]. Fig. 2.4 illustrates a simple example of MDP with three states and two actions. Especially, the MDP with a finite number of states is called finite MDP (FMDP). Most RL problems are categorized as FMDP. In MDP, the state transition follows Markov property; the next state only depends on the current state (not affected by past states). Therefore, the next state shows conditional independence to states and selected actions in the past. The MDP is recognized as the extended version of the Markov chain [110]

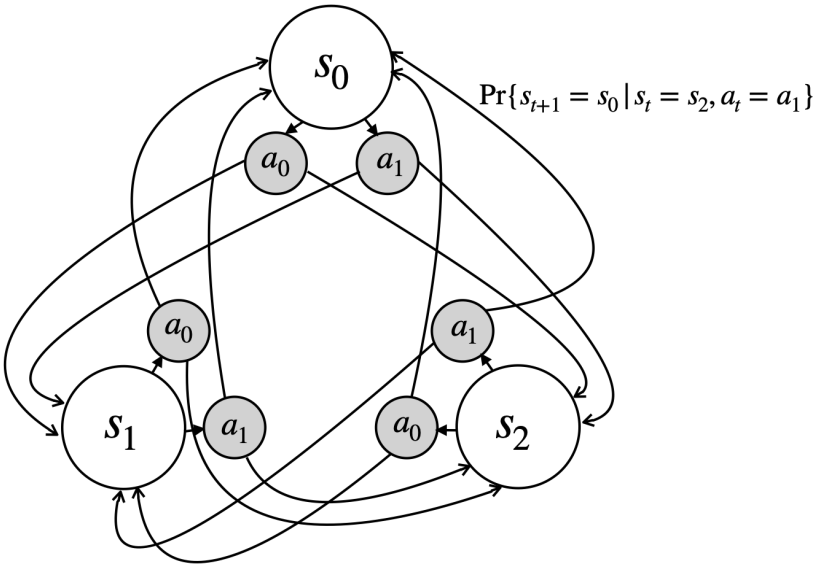


Fig. 2.4. Simple example of Markov decision process.

with the agent's action and reward that motivate the action. The FMDP is defined by the set of environmental states ($s \in S$), the set of agent's actions ($a \in A$), the probability of environmental transition, and the reward function. The probability of environmental transition is defined as follows:

$$\mathcal{P}^a_{ss'} = \Pr\{s_{t+1} = s' | s_t = s, a_t = a\} \tag{2.2}$$

where t is the index of the timestep. In addition, the expectation of reward is represented as follows:

$$\mathcal{R}^a_{ss'} = \mathbb{E}\{r_{t+1} | s_t = s, a_t = a, s_{t+1} = s'\}. \tag{2.3}$$

$\mathcal{P}^a_{ss'}$ and $\mathcal{R}^a_{ss'}$ are important factors to characterize the FMDP. The objective of MDP (or RL problem) is defined as maximizing long-term cumulative rewards.

Policy

The policy decides how the agent behaves in each timestep. Formally, the policy is defined as the mapping from the observed state of the environment to the agent's action. In the psychological field, the role of policy corresponds to the *stimulus-response rules* or *associations*. In some cases, the policy is represented by a mathematical function or lookup table, in others it also includes additional calculations such as a search process.

Reward function

The reward function detects the pair of an environmental state and an agent's action, then returns the reward signal which is a scalar value. The reward signal indicates how the selected action suppose to the environmental state is desirable to achieve the task. The goal

of the RL agent is to find the policy that maximizes cumulate reward in the task period. In biological systems, the reward can be analogous to the experiences of pleasure or pain.

Value function

In the RL field, the *value* is defined as the expectation of cumulative reward when the agent is in the state s and follows the policy π. In contrast to the reward function that shows the immediate evaluation, the value function indicates the long-term expectation. Formally, the value function is represented as follows:

$$V^{\pi}(s) = \mathbb{E}_{\pi}\{R_t | s_t = s\} = \mathbb{E}_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s\right\} \tag{2.4}$$

where the π is the agent's policy, and $R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$ (called *returns*) is defined as the function of the reward sequence. The γ is the *discount rate* parameter that decides how the agent takes into account the future reward at the current timestep ($\gamma = 0$ means the agent only focuses on the immediate reward). The $V^{\pi}(s)$ is called the *state-value function for policy* π. Similar to $V^{\pi}(s)$, another value function is formalized as follows:

$$Q^{\pi}(s, a) = \mathbb{E}_{\pi}\{R_t | s_t = s, a_t = a\} = \mathbb{E}_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s, a_t = a\right\}. \tag{2.5}$$

The $Q^{\pi}(s, a)$ also take into account the selected action a with the state s. The $Q^{\pi}(s, a)$ is called as the *action-value function for policy* π.

Generally, the values of $V^{\pi}(s)$ and $Q^{\pi}(s, a)$ are estimated based on the rewards that the agent iteratively experienced. For arbitrary state s and policy π, the value function satisfies the recursive form as follows:

$$\begin{aligned} V^{\pi}(s) = \mathbb{E}_{\pi}\{R_t | s_t = s\} &= \mathbb{E}_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s\right\} \\ &= \mathbb{E}_{\pi}\left\{r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | s_t = s\right\} \\ &= \sum_{a} \pi(s, a) \sum_{s'} \mathcal{P}^{a}_{ss'} \left[\mathcal{R}^{a}_{ss'} + \gamma \mathbb{E}_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | s_{t+1} = s'\right\}\right] \\ &= \sum_{a} \pi(s, a) \sum_{s'} \mathcal{P}^{a}_{ss'} [\mathcal{R}^{a}_{ss'} + \gamma V^{\pi}(s')]. \end{aligned} \tag{2.6}$$

Eq. 2.6 is called as *Bellman equation* for $V^{\pi}(s)$. This equation means the value of state s is equal to the summation between the expected reward and the expectation of discounted value in the next state. The RL solver is translated as finding optimal policies [*1] which

[*1] There may be more than one optimal policy.

obtain the maximum cumulative reward. For FMDP (i.e., if the number of states is not infinite), the optimal policies π^* can be defined by the value functions. The π^* share the *optimal -state-value function* V^* as follows:

$$V^*(s) = \max_{\pi} V^{\pi}(s) = \max_{a} \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V^*(s')]. \tag{2.7}$$

Similarly, the *optimal action-value function* Q^* is also shared as follows:

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a) = \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma \max_{a'} Q^*(s',a')]. \tag{2.8}$$

These equations for the optimal value functions are called *Bellman optimal equations*. Furthermore, the V^* and Q^* satisfy the following relation:

$$V^*(s) = \max_{a} Q^*(s,a). \tag{2.9}$$

Categorizations for RL algorithms

In [145], Sutton and Barto noted the most fundamental categorization for RL: dynamic programming (DP), Monte Carlo method, and temporal-difference (TD) learning. Typically, DP methods assume that the agent has a precise environment model. This setting is a mismatch to the many real-world problems, while DP supplies a lot of important theoretical aspects. Monte Carlo method uses the *experience* of the agent for training instead of the perfect environment model. TD learning involves both features of DP and Monte Carlo method; "update estimates based on other estimated values (i.e., use *bootstrapping*)" of DP and "not use a perfect environmental model" of Monte Carlo method. TD learning method is the hottest topic in the RL field; this thesis also employs the RL algorithm from TD learning. The following part exemplifies the TD learning concept on the value function update with the comparison to the Monte Carlo method. Both Monte Carlo method and TD-learning update the estimated $V(s_t)$ using an obtained experience. Typically, Monte Carlo method uses the following value update:

$$V(s_t) \leftarrow V(s_t) + \alpha[r_t - V(s_t)] \tag{2.10}$$

where r_t corresponds to the obtained reward at timestep t. On the Monte Carlo method, the reward history (r_1, r_2, ...) is available at the end of the episode (i.e., unit task period). Therefore, the value update has an episodic interval. In contrast, the TD-learning employs estimated $V(s_{t+1})$ and immediate reward r_{t+1} for updating $V(s_t)$. TD(0), the simplest TD-learning method use the following update:

$$V(s_t) \leftarrow V(s_t) + \alpha[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)]. \tag{2.11}$$

Based on this mechanism (temporal-difference-based update), TD learning realizes a relatively quick training process.

In addition to the most fundamental categorization, the rest of this section describes useful terminology for the RL. The features in the following part are important to characterize the algorithms in the RL (especially, TD-learning).

- **Value-based vs Policy-based**
  The value-based method mainly focuses on improving the value functions ($V(s_t)$, $Q(s_t, a_t)$). In this method, the policy is defined based on the value functions (e.g., greedy: deterministically select the action that maximizes $Q(s_t, a_t)$, ϵ-greedy: probabilistically select the action that maximizes $Q(s_t, a_t)$). Several algorithms such as Q-learning [157], and SARSA [144] are categorized into the value-based method. On the other side, the policy-based method typically defines the parametrized policy separate from the value function. Based on the policy gradient [146], the agent behaviors are improved by directly updating policy parameters. Examples of the method are REINFORCE [158], and natural policy gradient [67]. In addition, the policy-based methods often employ the value function update beside the policy update. This style is called the *actor-critic* [145].
- **On-policy vs Off-policy**
  On-policy and off-policy methods are distinguished whether the value function updates are based on the current policy or not. SARSA [144] is categorized into the on-policy method; the off-policy method includes Q-learning [157]. The rest part exemplifies the difference between on-policy and off-policy methods by using the SARSA and Q-learning. On the SARSA algorithm, the Q-function update is represented as follows:

$$Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma Q(s', a') - Q(s, a)]. \tag{2.12}$$

  Contrary, the Q-function update in Q-learning is represented as follows:

$$Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]. \tag{2.13}$$

  On the RL algorithms, the agent experiences are sampled as the set of (s: current state, a: selected action, r: obtained reward, s': next state, I: additional information(typically termination for task period)). Note that the $Q(s', a')$ in Eq. 2.12 requires a' following the current policy, while the $\max_{a'} Q(s', a')$ in Eq. 2.13 not depends on the current policy. This difference categorizes the SARSA and Q-learning into on-policy and off-policy, respectively. In general, the off-policy algorithms are good at sampling efficiency for agent experiences because the past experiences which not follow the current policy are reusable. The on-policy algorithms are less than the on-policy for experiences sampling efficiency, while they are good at stable training.

2.2.2 Deep Reinforcement Learning

In recent years, deep reinforcement learning (DRL) which is a combination of RL and deep neural networks (DNNs) has received a lot of attention from academies and industries. Typically, DRL employs DNNs as the function approximator for $V(s)$, $Q(s, a)$, and agent policy π. The rest of this section briefly describes the history of DRL.

In 2012, AlexNet [80] recorded a remarkable improvement in the image classification task. This achievement has started the era of *deep learning* [46, 83] which is an artificial intelligence field related to DNNs. In 2015, Mnih et al. propose the deep-Q-network (DQN) [100, 101] which is the method of approximating Q-function by DNNs. DQN has much attention due to the performances on computer games of Atari 2600; performances are equal or superior to the human players. Consequently, several improved techniques are proposed such as double-DQN[153], dueling network[156], general reinforcement learning architecture[105], and prioritized experience replay[127]. In 2017, several improving techniques are combined as the Rainbow algorithm[54], and it outperform the former methods. The Ape-X algorithm [58] in 2018 uses a distributed sampling mechanism and shows significant improvement in the performance of game AI. Finally, the Agent57 algorithm [2] recorded super-human performances in all of the Atari 57 games. These are the brief history of value-based methods which are mainly used for discrete action space.

On the other hand, the policy-based method of DRL started from the deterministic policy gradient (DPG) algorithm [135] in 2014. This algorithm became the base of some famous algorithms, such as deep deterministic policy gradient (DDPG) [86], soft actor-critic (SAC) [50], and twin-delayed DDPG (TD3) [40]. Another flow originated from the natural policy gradient algorithm [67]. In 2015 and 2017, Schulman et al. proposed trust region policy optimization (TRPO) [128] and proximal policy optimization (PPO) [129], respectively. The PPO algorithm becomes one of the most popular DRL algorithms. In contrast to the value-based methods, the policy-based methods are good at continuous control problems due to the independent-defined policy from the value function. This thesis also employs a policy-based method for a multi-legged robotic swarm. Typically, the benchmarks and application of continuous control become agent locomotion problems including multi-legged robots [53, 81, 97, 118]. In addition, both value-based and policy-based methods were utilized to train controllers of robotic swarms [60, 161], where there are relatively few examples. This thesis shows the first example of a combination of controlling multi-legged robots and robotic swarm on DRL.

2.3 Conclusions

This chapter presented a brief introduction to automatic design methods in swarm robotics. The automatic design methods are divided into two main domains: evolutionary robotics

and reinforcement learning. Evolutionary robotics is a common approach; typically, artificial neural networks are employed as the robot controller. In addition, this approach also succeeded in the design of gait for a multi-legged robot. Contrary, reinforcement learning is a more minor approach for robotics swarms, while recent developments are remarkable. The reinforcement learning also succeeded in designing locomotions with single-agent scenarios.

This thesis employs both approaches for the automatic design of controllers for a multi-legged robotic swarm: the hybridization problem of designing gait and collective behavior. Chapters 3 to 6 show that the evolutionary robotics approach successfully designs collective behavior of a multi-legged robotic swarm in several task scenarios. Additionally, Chapter 7 shows that reinforcement learning becomes an alternative to evolutionary robotics.

Chapter 3

Experimental Study on Generating Collective Step-climbing Behavior

This chapter presents an experimental study on generating collective step-climbing behavior of a multi-legged robotic swarm. The multi-legged robots can show physical interactions in three-dimensional space by using their legs and bodies. Therefore, the robotic swarm is expected to generate novel collective behaviors inspired by the self-assembly of army ants. In this chapter, a robotic swarm aims to achieve a collective step-climbing task; robots utilize other robots as stepping stones to climb over a step that is too high for a single robot. The robot systems for climbing other robots are discussed in self-assembling robots or modular robots [64, 92, 112, 121]. In most of these studies, connection rules between robots are determined by human designers; unit robots tend to have a simple shape and show relatively low movability. Contrary, multi-legged robots are good at the movability of a unit robot. Thus, they are expected to operate in exploring or transporting tasks. On the other hand, connection rules between robots (i.e., how to use legs on the other robot) are unpredictable for human designers.

In this study, the evolutionary robotics approach is employed to design a robot controller. A huge number of trial-and-error processes with parallel calculation become a promising way to design complicated control software. As mentioned in Chapter 1, the controller manages two types of actions: the basic gait for moving in the environment and the gait for climbing the step. In this study, the basic gait is realized by a central pattern generator (CPG) which is a common option for designing gaits of multi-legged robots [61, 88, 106]. The controller of a robotic swarm is developed by connecting an artificial neural network (ANN) to the hand-tuned CPG. The synaptic weights of ANN are optimized by evolutionary computation

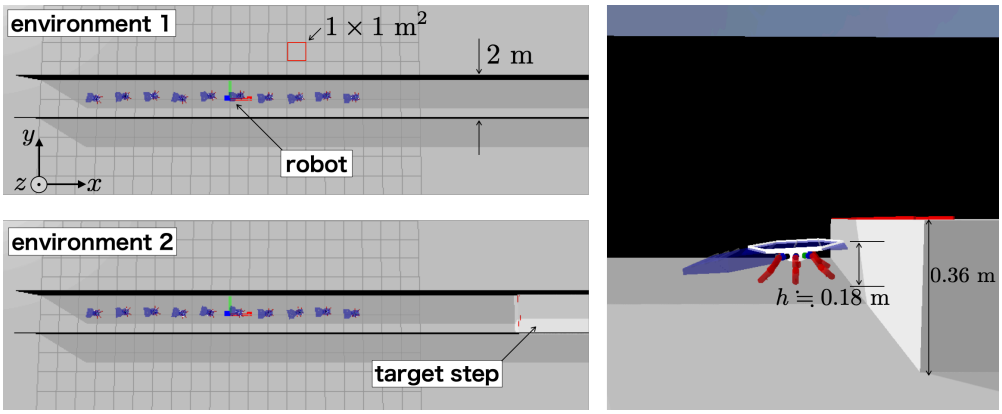


Fig. 3.1. Experimental environments. Environment 1 (Env-1) has no step within the environment, while environment 2 (Env-2) has a step with a height of 0.36 m.

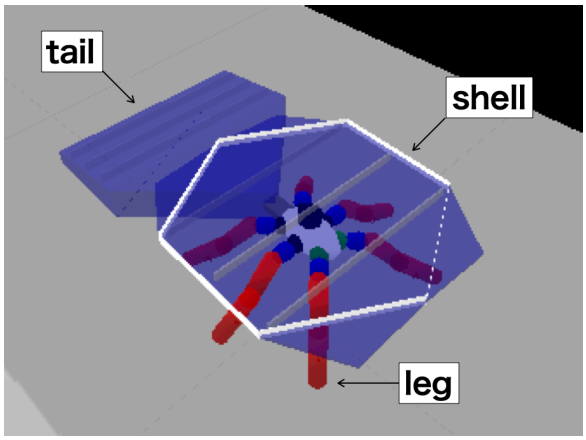


Fig. 3.2. Snapshot of the multi-legged robot.

to that the output of ANN affects the CPG signal and generates a collective behavior. The experiments are conducted in computer simulations. In addition, the obtained controllers are analyzed to check which sensor inputs are critical to achieving the task.

The remainder of this chapter is organized as follows. Section 3.1 describes the settings of experiments, and Section 3.2 presents methods for designing the robot controller. Section 3.3 shows the results obtained from the experiments. Section 3.4 shows the discussion about the obtained behavior of robots. Finally, Section 3.5 concludes this chapter.

3.1 Settings of Experiments

The collective step-climbing task requires the robots to climb a step, which is too high for a single robot to climb up. The robots utilize other robots as stepping stones for achieving the task. The experiments are carried out in computer simulations with the PyBullet physics engine [22].

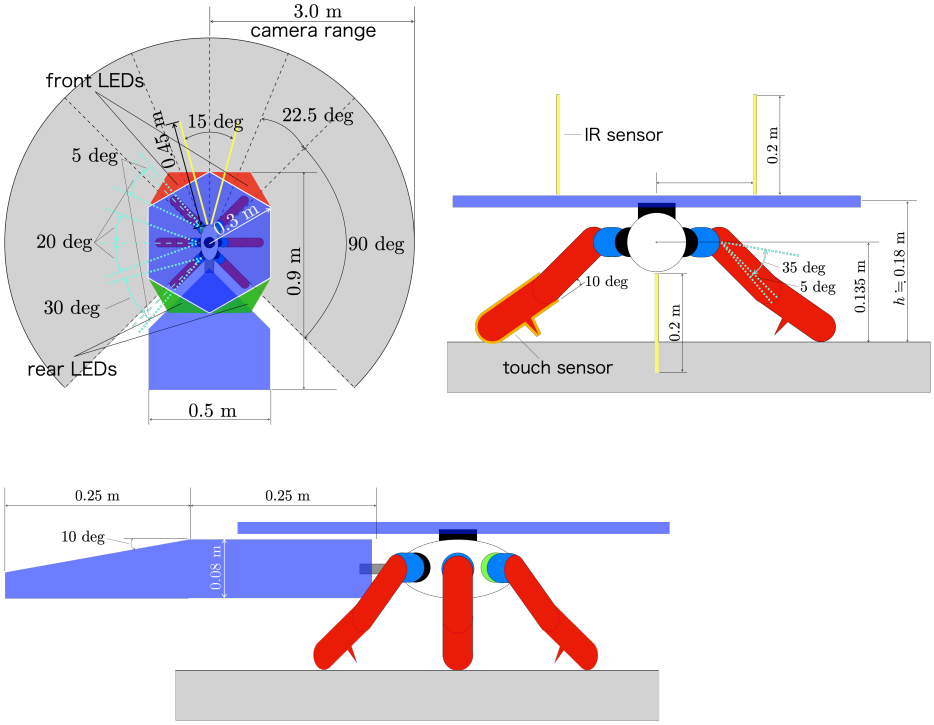


Fig. 3.3. Settings of the robot. Cyan dotted lines indicate the movable range of the joint. The gray circular sector shows the visible range of the camera. Yellow lines are the sensor ranges of the distance sensors.

3.1.1 Task Settings

Fig. 3.1 shows the environmental settings used in the experiments. The evolution processes are executed in two different environments, i.e., *environment 1* (Env-1) and *environment 2* (Env-2). Env-1 has no step within the environment; the robotic swarm aims to walk in a specific direction and form a trail like ants. Robots are placed at the initial position in a random direction in Env-1. In Env-2, the step is placed within the environment. Robots have to utilize other robots as stepping stones to climb over the step. When the robot succeeded to climb the step, it will be repositioned at 5 m in front of the step and join the task again. The initial directions of the robots are determined randomly with a smaller variance compared with Env-1 to make the robotic swarm focuses on acquiring the step-climbing behavior. The height of the step is 0.36 m, which is almost twice as high as the robot.

3.1.2 Robot Settings

The snapshot of the multi-legged robot is shown in Fig. 3.2. Additionally, configurations of the robot are illustrated in Fig. 3.3. Each robot has a shell and tail parts to support

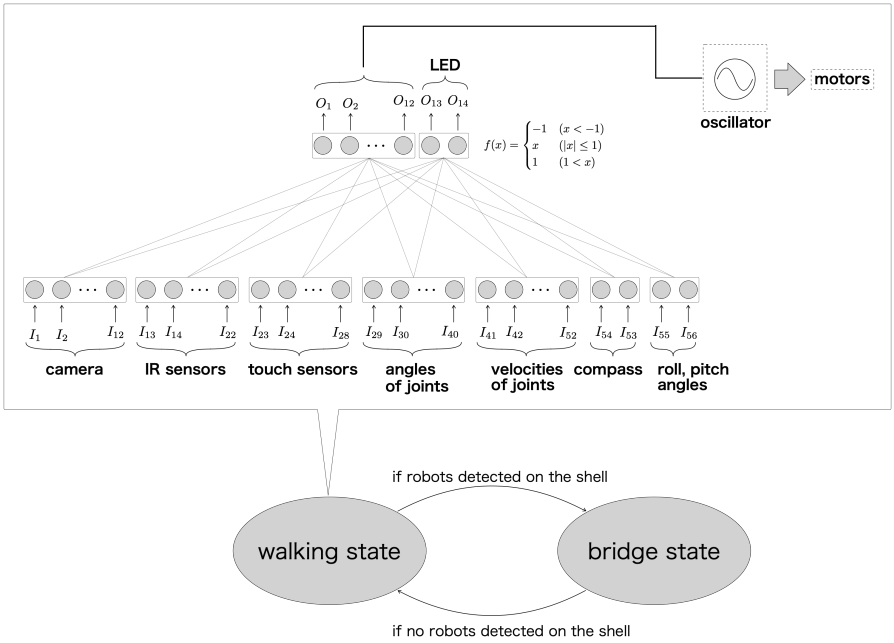


Fig. 3.4. Structure of the robot controller.

other robots to climb the step. In addition, the robot is equipped with LEDs, a camera, infrared (IR) sensors, and touch sensors. Two LEDs are attached to the front and rear parts of the shell. The camera has a visible range divided into six regions, as illustrated in Fig. 3.3. Each region can detect the colored LEDs on the front and back of the robot independently. The camera obtains the existence of colored LEDs by binary signals; in total, twelve binary signals are obtained from the camera. The value from the IR sensor is normalized into the range of [0,1]. Infrared sensors can distinguish between robots and other objects, and return values independently. Touch sensors are equipped at the end of each leg. Each touch sensor returns 1 when it detects collisions with other objects and 0 otherwise.

3.2 Methods

The evolutionary robotics approach is a promising method to design controllers for a robotic swarm. Typically, robot controllers in evolutionary robotics are represented by artificial neural networks. This study proposes a robot controller that combines the neural network with the simple oscillator (central pattern generator: CPG) and the finite state machine. The remaining part of this section describes the details of the controller and the evolutionary algorithm.

Table 3.1. Settings of the genetic algorithm.

Parameter	Value
total generations	1000
population size	50
tournament size	2
elite size	1
scaling parameter σ	0.02

3.2.1 Controller

The robot controller is illustrated in Fig. 3.4. The controller consists of three parts; the finite state machine, the neural network, and the oscillator. The finite state machine is composed of two states: the bridge state and the walking state. The bridge state is designed based on [92], and it is originally inspired by the behavior of ants in natural systems. In the bridge state, the robot stops walking to act as a stepping stone with keeping the body posture.

The walking state is composed of the neural network and the oscillator. The neural network is used to process signals from sensors. The input layer of the neural network consists of 56 neurons. Inputs are obtained from a camera, distance sensors, touch sensors, angles and velocities of joints, roll and pitch angles of the torso, and the compass. The output layer has 14 neurons. Twelve of them are involved in determining target angular velocities of joints, and the remaining two control the activation of LEDs. The oscillator is tuned to make the robot walk by tripod gait. The outputs of the neural network are multiplied by the oscillator signal to determine the target angular velocities of joints.

3.2.2 Evolutionary Algorithm

The evolutionary algorithm optimizes the synaptic weights of the neural network. The mutation-based genetic algorithm is employed, which is designed based on [142]. The mutation is represented by the following equation:

$$\boldsymbol{\theta}^n \leftarrow \boldsymbol{\theta}^{n-1} + \sigma\boldsymbol{\epsilon} \tag{3.1}$$

where $\boldsymbol{\theta}^n$ is the synaptic weights of the neural network in the nth generation, $\boldsymbol{\epsilon}$ is a noise vector sampled from a normal distribution, and σ is the scaling parameter that takes a scalar value. The parent controllers are selected using the tournament selection. Additionally, the elite selection is employed to reserve the controller that obtained the highest fitness value within the population. Table 3.1 summarizes the parameter settings of the evolutionary algorithm.

3.2.3 Fitness Function

The controller is evaluated by fitness function which is based on the performance of a robotic swarm. The fitness function consists of three parts; the first part is for walking in a specific direction, the second part is for following the other robots, and the third part is for climbing up the step. The first part is represented by the following equations:

$$fitness_1 = \max\left(\sum_i^{N_r} \frac{1}{c_1 + |\boldsymbol{x_t} - \tilde{\boldsymbol{x}}_{\boldsymbol{i}}|^\alpha} - c_2, 0\right) \cdot \delta_1 \cdot \delta_2 \tag{3.2}$$

$$\tilde{\boldsymbol{x}}_{\boldsymbol{i}} = (\min(x_i, S),\ y_i,\ z_i)$$

where N_r is the number of robots, $\boldsymbol{x_t}$ is the vector of the target point, and $\tilde{\boldsymbol{x}}_{\boldsymbol{i}}$ is the position vector of the ith robot. The position vector $\tilde{\boldsymbol{x}}_{\boldsymbol{i}}$ has the upper limit of S in the x coordinate. The c_1 is the constant value to prevent a division by zero. The c_2 is the constant value for the pseudo-linearization of the inverse proportionality. This function will make robots walk in a specific direction by setting $\boldsymbol{x_t}$ as sufficiently far from the initial positions of robots. The δ_1 and δ_2 are supplemental parts to realize the realistic ant-like gait. These values are calculated based on the cosine similarity between the ideal and actual states of robots. The δ_1 is set for the robots to walk forward, and the δ_2 makes robots move each leg equally.

The second part is for following other robots, which is represented by the following equations:

$$fitness_2 = \sum_t^T \sum_i^{N_r} f_{2,t,i} \tag{3.3}$$

$$f_{2,t,i} = \begin{cases} 1 & \text{if the } i\text{th robot detects LEDs in front two regions} \\ & \text{of the camera at the timestep } t \text{ and } x_i > I \\ 0 & \text{otherwise} \end{cases}$$

where T is the timesteps per generation and I is the threshold for activating this fitness function. This part is activated when the x coordinate of the ith robot is higher than I.

The third part is the fitness for climbing the step, which is defined as follows:

$$fitness_3 = \sum_i^{N_r} f_{3,i} \tag{3.4}$$

$$f_{3,i} = \begin{cases} 1 & \text{if } x_i > x_{step} \\ 0 & \text{otherwise.} \end{cases}$$

where x_{step} is the threshold to regard that the ith robot has climbed up the step.

The total fitness value is calculated by the weighted summation of three parts, which is described as follows:

$$Fitness = \frac{1}{M} \sum_i^M \sum_j^3 K_j \cdot fitness_j \tag{3.5}$$

where M is the number of evaluations and K_j is the constant value for scaling the corresponding part.

Table 3.2. Parameter settings in Env-1 and Env-2. Bold font is used to emphasize the changes in the two environments.

Parameter	Environment 1	Environment 2
timesteps per generation	1000	**800**
N_r	10	10
c_1	1.0	1.0
c_2	9.895×10^{-2}	9.895×10^{-2}
α	0.5	0.5
S	45.0	45.0
I	20.0	**12.5**
x_{step}	16.5	16.5
K_1	50.0	50.0
K_2	2.0×10^{-3}	$\mathbf{4.0 \times 10^{-4}}$
K_3	0.0	**3.0**
M	1	**3**

3.2.4 Experimental Conditions

The process of artificial evolution is divided into two phases. First, the robot controllers are evolved in Env-1. Next, evolved controllers are set as the initial population and re-evolved in Env-2. The parameter settings for Env-1 and Env-2 are shown in Table 3.2. The parameter settings in Env-2 are adjusted to address the step-climbing task.

In Env-1, two types of controllers will be evolved: one with the oscillator and one without the oscillator. After that, four experiments with different controller settings are conducted in Env-2 to compare the performance. The settings are described as follows:

Setting A: The standard settings with using all of the parts in the controller.
Setting B: The controller without the oscillator to test the effect of the oscillator.
Setting C: The controller with only the walking state and without the bridge state.
Setting D: The supplementary experiment, which is conducted using one robot to confirm the difficulties of the task.

Controllers with the oscillator evolved in Env-1 are used as the initial population at setting A, C, and D. Controllers without the oscillator are used at setting B. Five trials are performed for each setting.

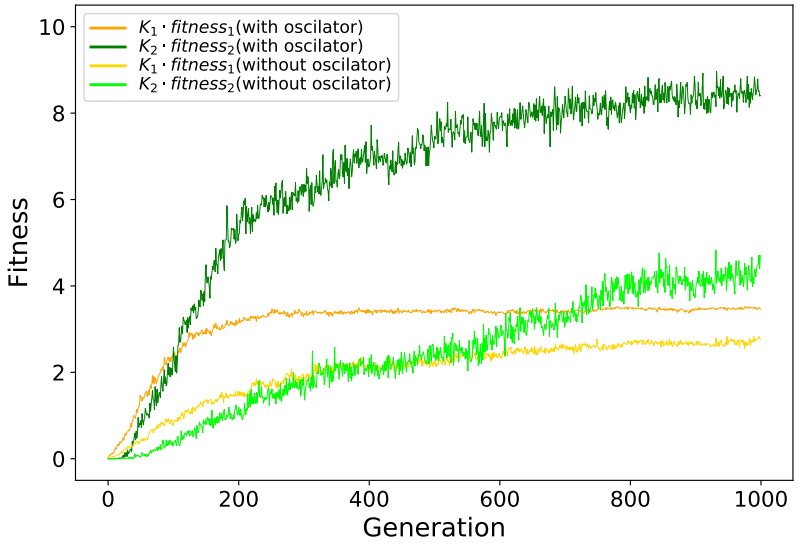


Fig. 3.5. Fitness of the best individual across generations in Env-1. Lines indicate the average over five trials.

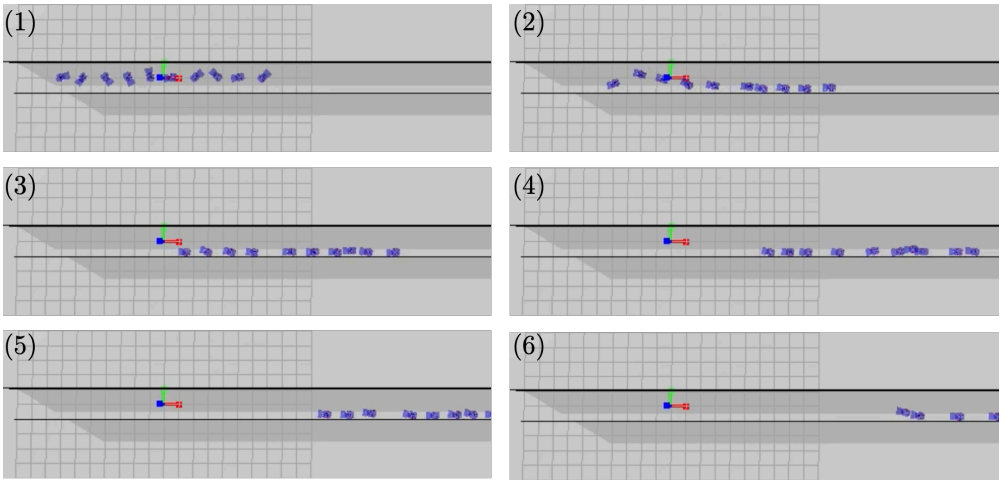


Fig. 3.6. Example of collective behavior in Env-1.

3.3 Results

The fitness transitions in Env-1 are shown in Fig. 3.5. Each line indicates the average of the best individual across generations over five evolutionary trials. The figure shows that the oscillator contributed to accelerating the evolution and improving the final performance. Fig. 3.6 shows the example of generated behavior in Env-1 by using the best-evolved controller. The multi-legged robotic swarm could keep a line formation and move in a direction along the x-axis.

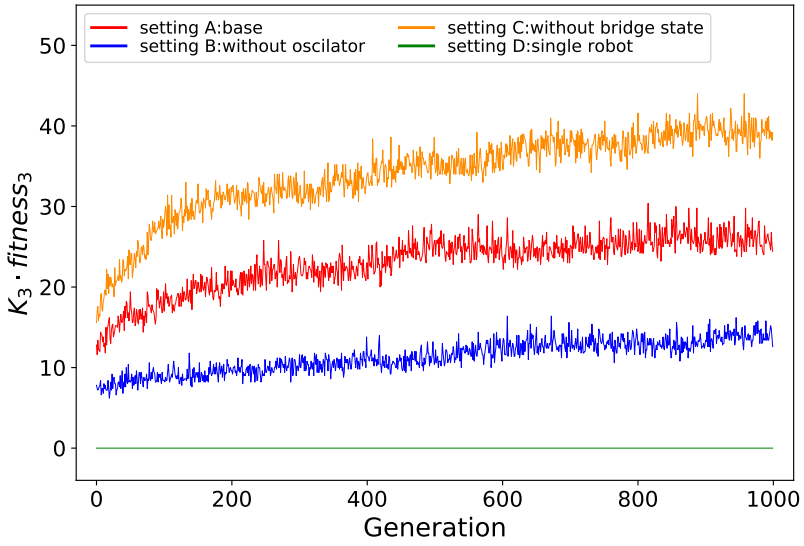


Fig. 3.7. Fitness of the best individual across generations in Env-2.

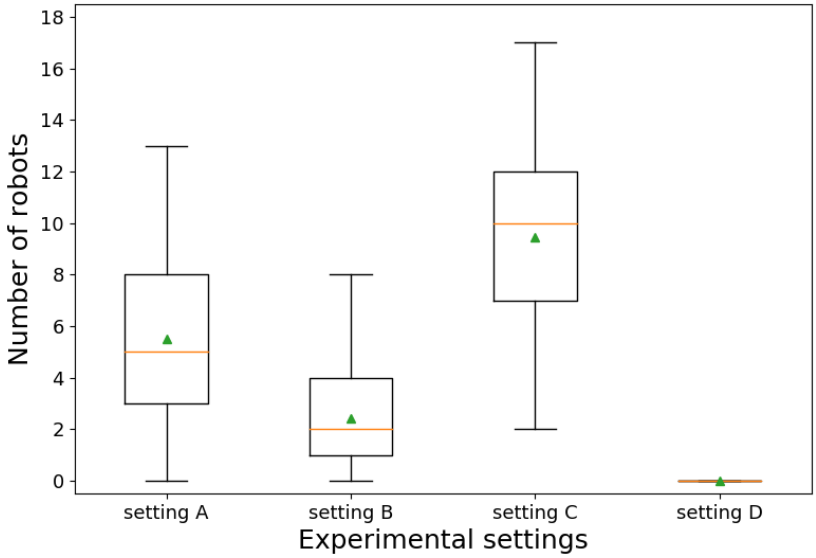


Fig. 3.8. Box plots for the number of robots that have climbed the step over 100 trials. The best-evolved controller in each experimental setting is evaluated.

The fitness transitions in Env-2 are shown in Fig. 3.7. Each line indicates the average of the best $K_3 \cdot fitness_3$ value over five evolutionary trials. Additionally, the re-evaluation using the best-evolved controller was performed for 100 trials. The results of the re-evaluation are shown in Fig. 3.8. The results show that the oscillator also improves the performance in Env-2. Moreover, the controller without the bridge state scored the highest fitness values. It seems that the bridge state is not contributing to improving performance. It can be

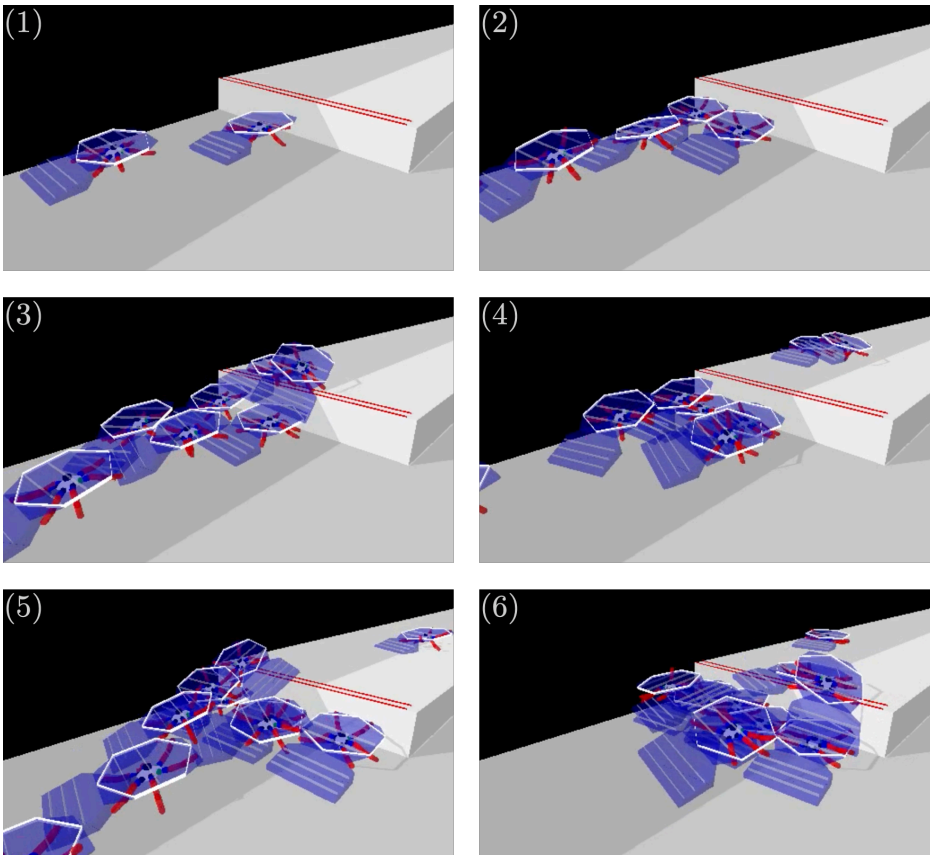


Fig. 3.9. Example of collective step-climbing behavior generated in Env-2.

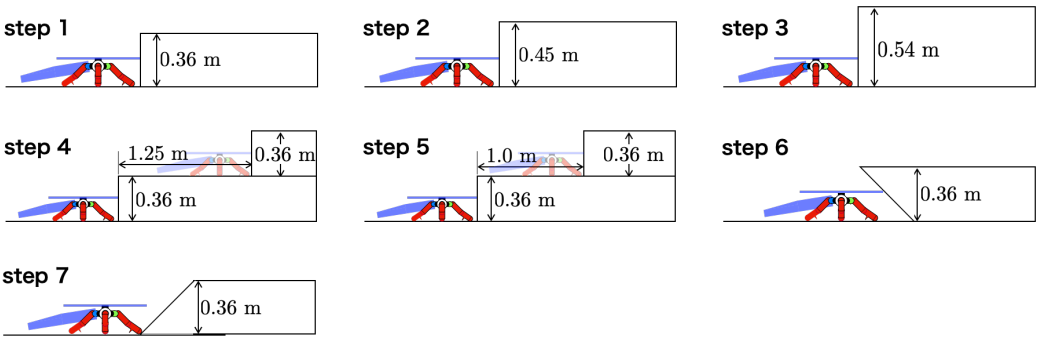


Fig. 3.10. Environmental settings for the scalability test. The steps have a different height or shape from the original one (step 1).

assumed that the task scenario is relatively simple and the robots did not need to take explicit altruistic behavior of becoming a stepping stone. The robot in setting C kept walking toward the step, and therefore, it becomes a more stable and higher stepping stone by pressing its shell against the step. On the other side, the results of setting D show that the $K_3 \cdot fitness_3$ is always zero. This means the robot could not achieve the task alone. Fig. 3.9 shows the example of generated behavior in Env-2 by using the best-evolved controller of

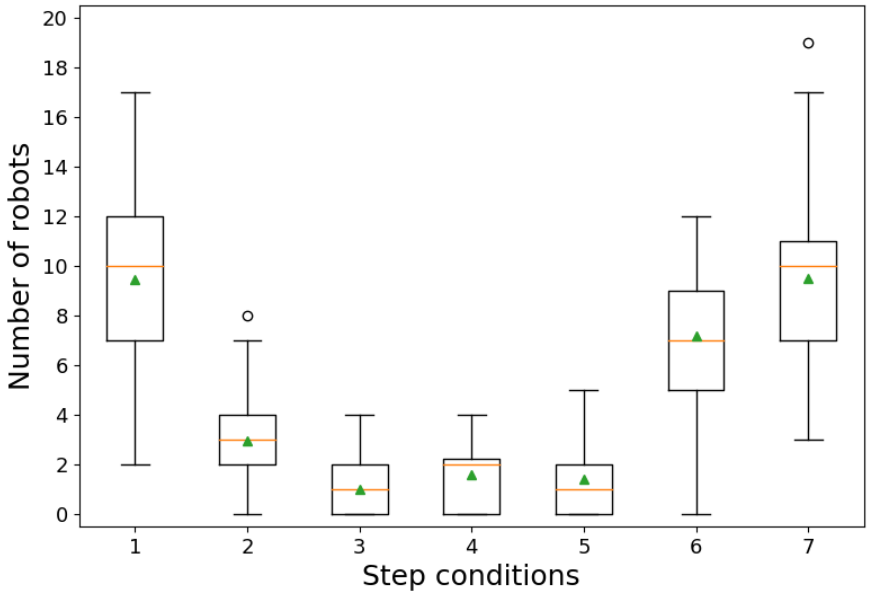


Fig. 3.11. Box plots for the number of robots that have climbed the step in each step condition. The best-evolved controller in setting C is evaluated over 100 trials.

Setting C. The multi-legged robots could climb the step using the robots in front of the steps as stepping stones.

Additionally, a flexibility test for the robotic swarm was conducted with different heights and shapes of steps. The best-evolved controller of Setting C was applied to six different steps. The settings of the steps are shown in Fig. 3.10. In steps 2 and 3, the height is increased to 0.45 and 0.54 m, respectively. Steps 4 and 5 are set as advanced tasks compared to the standard settings in step 1. In steps 6 and 7, the shape is changed to have an acute and obtuse angle. Fig. 3.11 shows the results obtained in the flexibility test. The performance in each step was evaluated 100 times, which is the same as Fig. 3.8. The robotic swarm succeeded in climbing up the step in all environments. The robotic swarm showed low performance in steps 2, 3, 4, and 5. These results showed that the height of the step seriously affects the performance of the robotic swarm. The results also showed that the robotic swarm is robust to the shape of the step.

3.4 Discussion

For further understanding of the obtained behaviors, evolved controllers are tested in Env-2 with deactivating sensors on robots. By performing this test, we can predict the contributions of each sensor in the task and analyze actions obtained by robots. A total of eight settings are defined as follows:

Setting I : The default setting (with activating all sensors).

Table 3.3. The number of controllers in which the performance significantly decreased from setting I. There are 10 controllers for each evolution trial, with a total of 50 controllers compared for each setting.

	Settings						
Evolution Trials	*II*	*III*	*IV*	*V*	*VI*	*VII*	*VIII*
Trial1	9	0	7	0	1	1	1
Trial2	5	0	1	6	0	0	0
Trial3	9	0	0	9	1	0	0
Trial4	3	1	1	3	2	1	1
Trial5	8	1	10	7	2	0	5
Total	34	2	19	25	6	2	7

Setting II : Deactivate the camera.
Setting III : Front IR sensors can detect only robots.
Setting IV : Front IR sensors can not detect robots.
Setting V : The bottom IR sensor can detect only robots.
Setting VI : The bottom IR sensor can not detect robots.
Setting VII : IR sensors on the shell can detect only robots.
Setting VIII : IR sensors on the shell can not detect robots.

When the sensor is deactivated, it returns a signal as if it is not detecting the object. The input neurons that correspond to the deactivated sensors will receive zero values. In each test setting, controllers evolved with Setting C are tested 100 times. As mentioned in section 4, evolution processes were conducted five times. Controllers with the top 10 fitness values are selected from each trial, in a total of 50 controllers are tested.

The results of the test are shown in Table 3.3. This table summarizes the number of controllers out of 50 in which the performance significantly decreased from setting I (Mann-Whitney U test, p-value <0.05). The results show that the performance was most affected by the camera; 34 out of 50 controllers significantly decreased performance when deactivating the camera. The second was setting V, with 25 out of 50 controllers showing a lower performance. Deactivating the bottom sensor will make the robot difficult to measure the distance between the floor and the body, which leads to the robots having lower body postures. In setting IV, 19 out of 50 controllers showed a significantly lower performance. Settings II and IV have a similar role by making the robot difficult to detect other robots in the front. Since setting II has the largest influence on performance, we focus on setting II and perform the additional

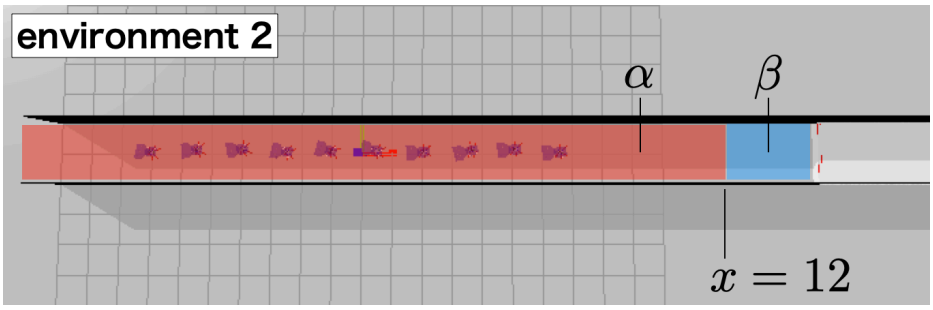


Fig. 3.12. The new environmental setting. The passage of Env-2 is divided into two regions, α and β.

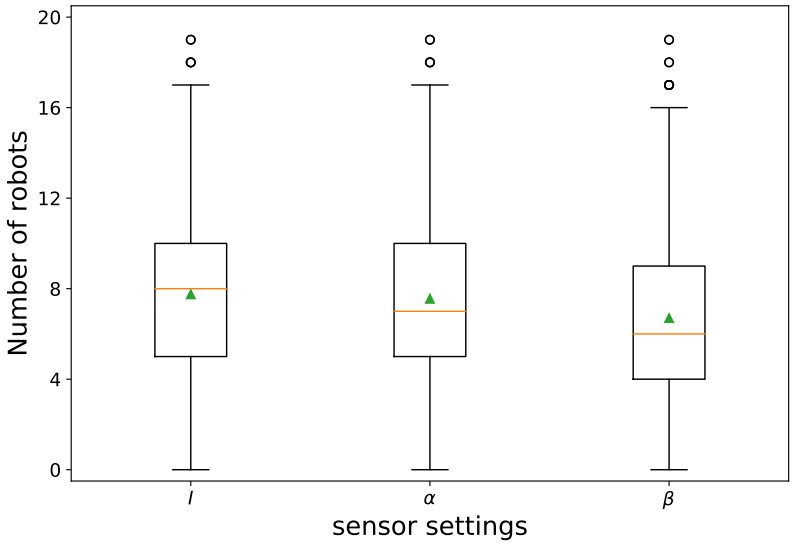


Fig. 3.13. Box plots for the number of robots that have climbed the step in setting α and β.

test.

One possible effect of deactivating the camera is that it becomes difficult for robots to form a line. Deactivating the camera may lead to increase collisions and overlaps between robots and decrease performance. To test this hypothesis, the new environmental setting is introduced, as shown in Fig. 3.12. This environment is divided into two regions, α and β. The new sensor settings are introduced considering the two regions, which are defined as follows:

Setting α: Deactivate the camera at the region α.
Setting β: Deactivate the camera at the region β.

These settings are designed for inspecting the importance of forming a line within the task.

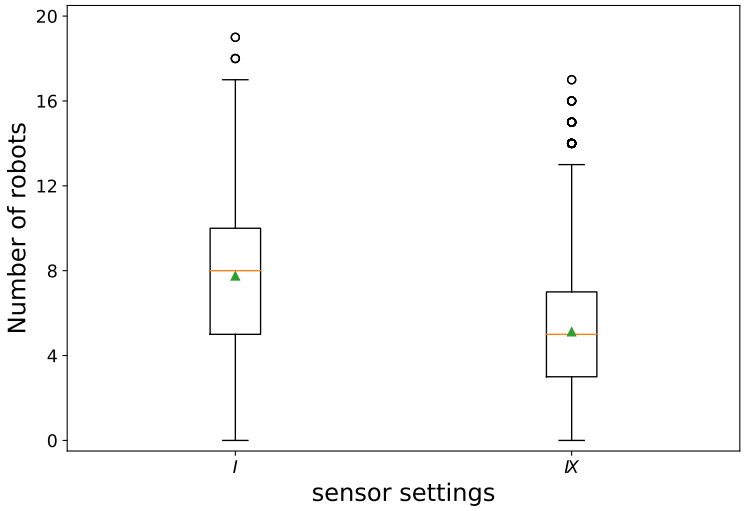


Fig. 3.14. Box plots for the number of robots that have climbed the step in setting *I* and *IX*.

If forming a line is important to the task, the score for setting α is likely to be lower than for setting β because region α is much longer than region β. A total of 50 controllers from setting C are applied for setting α and β, similarly to the previous test. The results of this test are shown in Fig. 3.13. The results show that the hypothesis is not appropriate, i.e., the detection of robots just in front of the step is more important for achieving the task. It can be assumed that the robots seem to be locating other robots in front of the step to utilize them as stepping stones.

Finally, the contributions of the robot embodiment and evolved controllers on the performance are analyzed. The additional sensor setting is introduced as follows:

Setting IX: Use Setting *II* – *VIII* together (deactivate the camera and all of the IR sensors).

This setting is designed to test the contribution of an embodiment because robots have to address the task by only walking in a specific direction. Similar to the other settings *I*–*VIII*, setting *IX* is applied for a total of 50 controllers from setting C. The results of the experiments using setting *IX* are shown in Fig. 3.14. The results show that the task could be achieved to some extent by only walking toward the step. However, the performance of climbing the step was decreased. This loss of performance shows that the evolved controller has contributed to the step-climbing behavior.

3.5 Conclusions

This chapter presented an experimental study on the collective step-climbing behavior of the multi-legged robotic swarm. The evolutionary robotics approach was employed to design the neural network within the robot controller. Computer simulations showed that the proposed method successfully achieves the step-climbing task. Additionally, the evolved controller was able to achieve the task with steps that are different heights and shapes. Furthermore, the analysis for obtained controllers showed that the performance of the robotic swarm depends on robot embodiment. However, the results also showed the robot controllers evolved to contribute to the performance.

Chapter 4

Neuroevolution Approach for Generating Collective Behavior of a Multi-Legged Robotic Swarm

This chapter focuses on a pure neuroevolution approach for a multi-legged robotic swarm. In Chapter 3, the proposed method employed CPG for the basic gait of a single robot. This method successfully established a hierarchical design of single robot action to collective behavior due to the simple mechanism of CPG. However, the gait itself became monotonous and follows a fixed cycle. In addition, if the robot settings (e.g., the movable range of joints, the friction coefficient of legs) are changed, the CPG needs to be re-tuned.

This chapter proposes a new method of pure neuroevolution: the basic gait of a single robot is also designed by a neural controller with an evolutionary robotics approach. Generally, a multi-legged robot shows enormous aspects of gaits due to the large degrees of freedom. Additionally, several gaits are not similar to natural creatures; they are may unsuitable to achieve the task by mimicking natural creatures. In this study, intuition-based fitness parts were designed to make robot gaits similar to legged animals. As Chapter 3, the experiment for designing collective behavior was conducted in computer simulation with the physics engine. The robotic swarm was evolved in the task of forming a line. The evolution was conducted with a hierarchical process: the first for evolving the gait of a single robot and the second for evolving collective behavior. The results showed that the neuroevolution approach successfully designed both a basic gait and collective behavior of a multi-legged robotic swarm.

The rest of this chapter is organized as follows. Section 4.1 presents experimental settings.

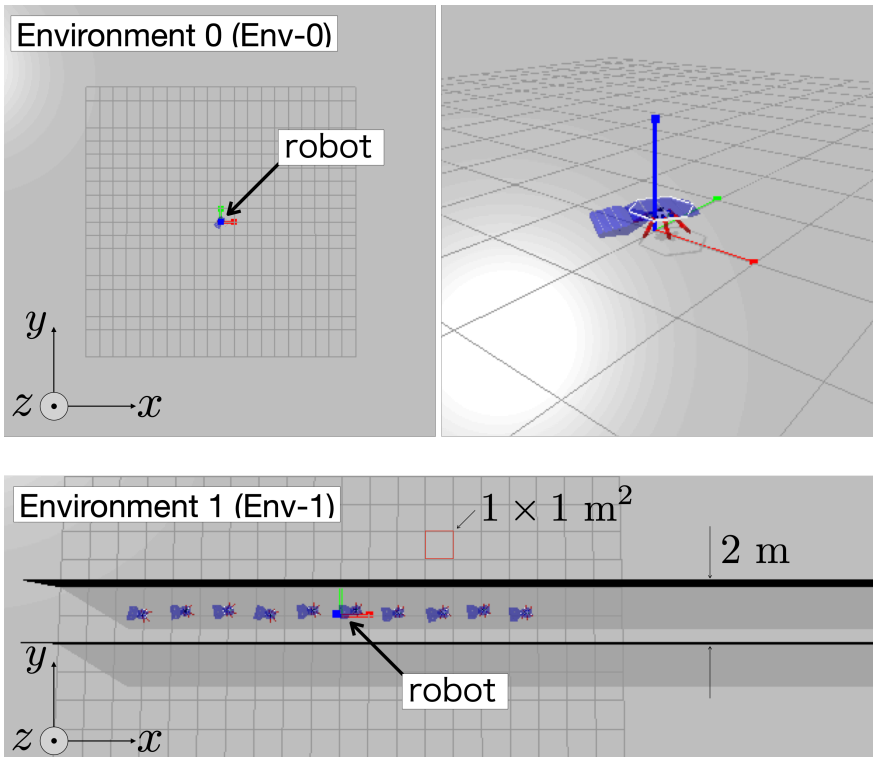


Fig. 4.1. The settings of the task environments. Environment 0 (Env-0) is the flat field that designs a gait of a single robot. Environment 1 (Env-1) has walls and evolves collective behavior using 10 robots.

Section 4.2 describes the details of the neuroevolution approach. Section 4.3 shows the results and discussion. Finally, Section 4.4 presents the conclusion.

4.1 Settings of Experiments

This section describes the experimental settings. The experiments in this study aim to generate collective behavior of a multi-legged robotic swarm based on the neuroevolution approach. A total evolution process is divided into two stages. The first stage is for designing a gait of a single robot. The second stage requires robots to form and keep a line while they are moving. As in Chapter 3, the experiments are carried out in computer simulations.

Fig. 4.1 shows the settings of the task environments. The evolution processes are executed in two environments, i.e., *environment 0* (Env-0) and *environment 1* (Env-1). Env-0 is the flat field represented by $z = 0$. In Env-0, a single robot is evolved to walk in a direction along with the x-axis. After the evolution in Env-0, the obtained controllers are re-evolved in Env-1 with ten robots. Env-1 has two walls that continue until $x = 45.0$. In Env-1, robots are evolved to follow other robots and form a line.

In addition, the robot configurations are illustrated in Fig. 4.2. The body structure of the

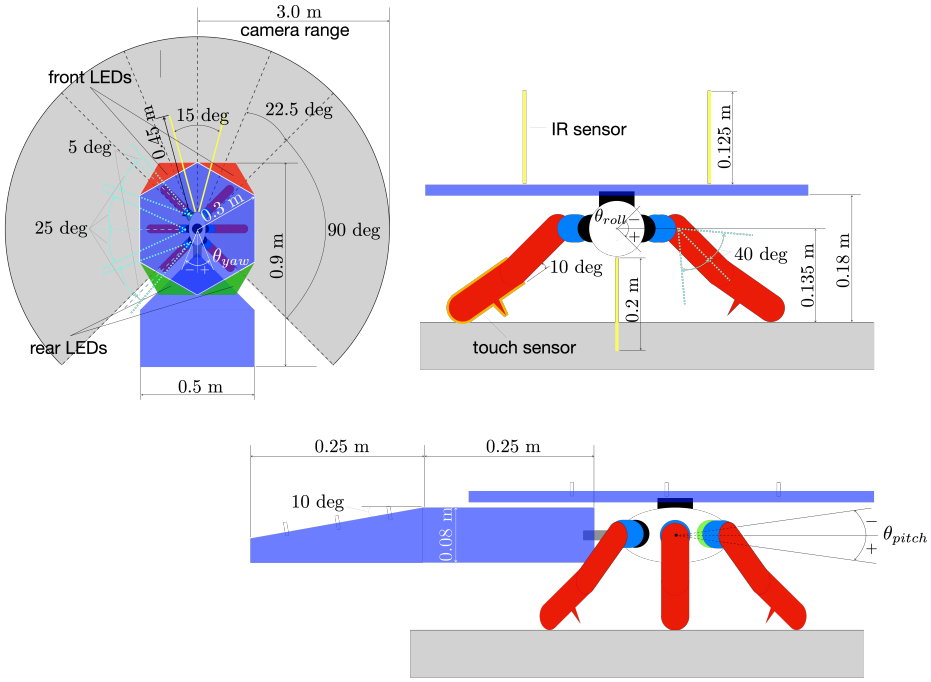


Fig. 4.2. Settings of the robot. The main changes from Chapter 3 are movable ranges of joints; they become wider for the better movabilities of robots. As in Chapter 3, cyan dotted lines indicate the movable range of joints. The gray circular sector shows the visible range of the camera. Yellow lines are the sensor ranges of the infrared sensors.

robot is similar to Chapter 3 except for the movable ranges of joints. The joints have wider movable ranges than in Chapter 3 for increasing the movability. The basic mechanism and role of sensors are also the same to Chapter 3.

4.2 Methods

The neuroevolution approach is employed to design the controller for a robotic swarm. The remaining part of this section describes the details of the controller and the evolutionary computation method.

4.2.1 Controller

The robot controller is illustrated in Fig. 4.3. The recurrent neural network with a single hidden layer is employed as the robot controller. The network has recurrent connections at the hidden layer and direct connections from the input layer to the output layer. The input layer consists of 56 neurons. Inputs are obtained from a camera, IR sensors, touch sensors, angles and angular velocities of joints, roll and pitch angles of the torso, and the compass. The output layer has 14 neurons. Twelve of them decide the target angular velocities of

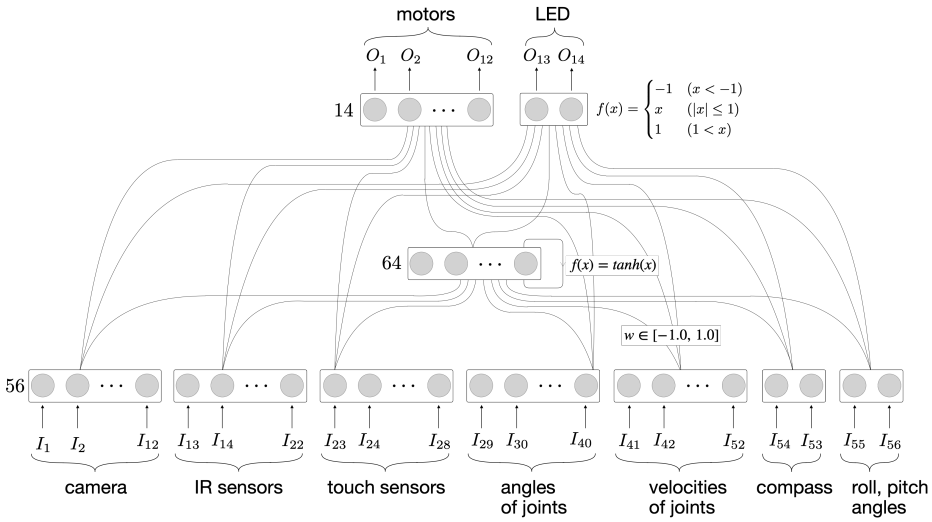


Fig. 4.3. Structure of the robot controller.

Table 4.1. Parameter settings of the (μ, λ)-ES.

Parameter	Value
Number of parents μ	20
Number of offspring λ	64
Initial mutation step size	0.005
Mutation step size	$\in [0.0005, 0.15]$
Terminate generation $G_{\max}$	1000
Proportional constants (c, c')	(0.5, 2.5)

joints, and the remaining two control the activation of LEDs.

4.2.2 Evolutionary Algorithm

A total of 9360 synaptic weights in the controller are optimized by the (μ, λ)-evolution strategies (ES)[30]. Algorithm 2 shows the pseudocode of the (μ, λ)-ES. In addition, parameter settings are summarized in Table 4.1. In several studies, the ratio of μ and λ is set as 1/7[30]. This experiment employs weaker selection pressure ($\mu/\lambda \approx 1/3$) to satisfy multiple objectives in the fitness function (described in 4.2.3). The parameter setups in Table 4.1 are determined based on preliminary experiments with Exp-0.

4.2.3 Fitness Function

The fitness function consists of two parts; $fitness_1$ for walking in a specific direction, and $fitness_2$ for following the other robots. The role of each fitness part is similar to Chapter 3,

Algorithm 2 Pseudocode of (μ, λ)-ES.

Input:

fitness function $F(\boldsymbol{x}) : \mathbb{R}^n \to \mathbb{R}$
number of parents μ
number of offsprings λ
constants of proportionality c, c'
terminate generation $G_{\max}$

Define:

$\tau = \frac{c}{\sqrt{2\sqrt{n}}}, \ \tau' = \frac{c'}{\sqrt{2n}}$

Initialize:

generation counter $G \leftarrow 0$
for $i \leftarrow 1,\ldots,\lambda$ **do**
    individual $\boldsymbol{\mathcal{X}}^{(i)} = \{\left(\boldsymbol{\theta}^{(i)}, \boldsymbol{\sigma}^{(i)}\right) \mid \boldsymbol{\theta}^{(i)}, \boldsymbol{\sigma}^{(i)} \in \mathbb{R}^n\}$
end for
initial population $\boldsymbol{\mathcal{P}}_G \leftarrow \{\boldsymbol{\mathcal{X}}^{(1)}, \ldots, \boldsymbol{\mathcal{X}}^{(\lambda)}\}$
fitness of individuals $\boldsymbol{F}_G \leftarrow \{F(\boldsymbol{\theta}^{(1)}), \ldots, F(\boldsymbol{\theta}^{(\lambda)})\}$

while $G < G_{\max}$ **do**
    Select μ individuals with the highest fitness from $\boldsymbol{\mathcal{P}}_G$
    for $i \leftarrow 1,\ldots,\lambda$ **do**
        Select the $\boldsymbol{\mathcal{X}}^{(i)}$ randomly from μ individuals
        $\boldsymbol{\sigma}'^{(i)} \leftarrow e^{\tau' \cdot N(0,1)} \cdot \boldsymbol{\sigma}^{(i)} \odot e^{\tau \cdot \mathcal{N}_n(0,I)}$
        $\boldsymbol{\theta}'^{(i)} \leftarrow \boldsymbol{\theta}^{(i)} + \boldsymbol{\sigma}'^{(i)} \odot \mathcal{N}_n(0, I)$
        $\boldsymbol{\mathcal{X}}'^{(i)} \leftarrow \{(\boldsymbol{\theta}'^{(i)}, \boldsymbol{\sigma}'^{(i)})\}$
    end for
    $\boldsymbol{F}_G \leftarrow \{F(\boldsymbol{\theta}'^{(1)}), \ldots, F(\boldsymbol{\theta}'^{(\lambda)})\}$
    $\boldsymbol{\mathcal{P}}_G \leftarrow \{\boldsymbol{\mathcal{X}}'^{(1)}, \ldots, \boldsymbol{\mathcal{X}}'^{(\lambda)}\}$
    $G \leftarrow G + 1$
end while

while several updates are introduced. Each fitness part is described below.

· $fitness_1$

The $fitness_1$ is the product of the base function and supplemental parts. The base function of $fitness_1$ is represented by the following equations:

$$f_1 = \max\left(\sum_i^{N_r} \frac{1}{c_1 + |\boldsymbol{x_t} - \tilde{\boldsymbol{x}}_{\boldsymbol{i}}|^\alpha} - c_2, 0\right) \tag{4.1}$$

$$\tilde{\boldsymbol{x}}_{\boldsymbol{i}} = (\min(x_i, S),\ y_i,\ z_i)$$

where N_r is the number of robots, $\boldsymbol{x_t}$ is the vector of the target position, and $\tilde{\boldsymbol{x}}_{\boldsymbol{i}}$ is the position vector of the ith robot. The f_1 has a similar form to the function used in Chapter 3. The $\tilde{\boldsymbol{x}}_{\boldsymbol{i}}$ has the upper limit of S in the x coordinate. c_1, c_2, and α are constant values. This function will make robots walk in a specific direction by setting $\boldsymbol{x_t}$ as sufficiently far from the initial positions of robots.

When only using f_1, the robot controller often shows *unnatural* gaits (such as backward walking, shaking the torso frequently, and using few legs) due to a large degree of freedom. To solve this problem, a total of five supplemental fitness parts ($\delta_1 \sim \delta_5$) are introduced; they design the robot's gait similar to natural organisms. δ_1 is for walking forward, which is represented by the following equation:

$$\delta_1 = K_{\delta_1} \cdot \max\left(\frac{1}{N_r}\sum_i^{N_r} \frac{\boldsymbol{f}\cdot\boldsymbol{v_i}}{|\boldsymbol{f}||\boldsymbol{v_i}|}, 0\right) \tag{4.2}$$

where K_{δ_1} is the constant value, $\boldsymbol{v_i}$ is the velocity vector of robot i, and $\boldsymbol{f} = (0, 1, 0)$ is the vector indicating the front direction on the local coordinate of robots. δ_1 is the function of cosine similarity for the $\boldsymbol{v_i}$ and the $\boldsymbol{f}$. The value of δ_1 increases when robots show forward walking.

δ_2 is for driving joints, which is represented as follows:

$$\delta_2 = K_{\delta_2} \cdot \frac{1}{N_r}\sum_i^{N_r} \frac{|\boldsymbol{\phi}_i|}{|\boldsymbol{\phi}_i| + c_3} \cdot \left(\frac{\boldsymbol{\phi}_i \cdot \mathbf{1}}{|\boldsymbol{\phi}_i||\mathbf{1}|}\right)^{\beta} \tag{4.3}$$

where K_{δ_2}, c_3, and β are constant values, $\boldsymbol{\phi}_i$ is the vector indicating the rotation angles of joints. The detail of the $\boldsymbol{\phi}_i$ is described in Appendix A. The $\mathbf{1}$ is a vector whose elements are all 1. The value of δ_2 increases when robots move each leg equally.

δ_3, δ_4, and δ_5 are represented by the following equation:

$$\delta_j = \frac{1}{N_r}\sum_i^{N_r}\left\{\min\left(\frac{c_{4,\delta_j}}{1 + c_{5,\delta_j}\chi_{i,\delta_j}^{\gamma_{\delta_j}}}, 1\right)\cdot a_{\delta_j} + b_{\delta_j}\right\},\quad j \in \{3, 4, 5\} \tag{4.4}$$

where c_{4,δ_j}, c_{5,δ_j}, γ_{δ_j}, a_{δ_j}, and b_{δ_j} are constant values. χ_{i,δ_j} is the scalar value that shows the robot state to be minimized. In δ_3, χ_{i,δ_3} is the amount of change in the roll and pitch angles from standard values. Therefore, δ_3 requires the robot to keep orientation. In δ_4, χ_{i,δ_4} is the amount of change in the bottom IR sensor from the standard value to keep the distance between the body and the floor. In δ_5, χ_{i,δ_5} is the number of times the direction of the rotating joint has changed to prevent vibrations on joints. The details of the χ_{i,δ_j} are also described in Appendix A. The maximum values of δs are arranged to balance the effects of each part. Appendix B summarizes the parameter values used in the experiment.

The $fitness_1$ is calculated by the following equations:

$$fitness_1 = f_1\delta_1\delta_2 \cdot g(f_1\delta_1\delta_2,\ \prod_{k=3}^{5}\delta_k,\ \rho) \tag{4.5}$$

$$g(x,\ y,\ \rho) = \begin{cases} y & \text{if } x \geq \rho x_{max} \\ 1 & \text{otherwise} \end{cases}$$

where ρ is the constant value, and x_{max} is the theoretical maximum of x. The function $g(x,\ y,\ \rho)$ is introduced for the incremental evolution based on fitness functions [104]. By using $g(x,\ y,\ \rho)$, it becomes easy to satisfy all requirements of δs.

· $fitness_2$

The $fitness_2$ is for following other robots, which is represented by the following equations:

$$fitness_2 = \sum_{t}^{T} \sum_{i}^{N_r} f_{2,t,i} \tag{4.6}$$

$$f_{2,t,i} = \begin{cases} 1 & \text{if the } i\text{th robot detects LEDs in front two regions} \\ & \text{of the camera at the timestep } t \text{ and } x_i > I \\ 0 & \text{otherwise} \end{cases}$$

where T is the timesteps per generation, and I is the threshold for activating this fitness function. This part is activated when the x coordinate of the ith robot is higher than I.

◇Total fitness

The total fitness value is calculated by the weighted summation of two parts, which is described as follows:

$$Fitness = \frac{1}{M} \sum_{i}^{M} \sum_{j}^{2} K_j \cdot fitness_j \tag{4.7}$$

where M is the number of evaluations and K_j is the constant value for scaling the corresponding part. The parameter values in the fitness function are summarized in Appendix B.

4.2.4 Experimental Setup

The experiment process is divided into two parts, i.e., *experiment 0* (Exp-0) and *experiment 1* (Exp-1). The rest of this section describes the details of each part.

· Exp-0

Exp-0 is conducted in Env-0. The purpose of Exp-0 is to evolve the gait of a single robot. Therefore, Exp-0 uses only $fitness_1$. To check the effect of supplemental fitness parts, three settings are introduced as follows:

Setting 0-A: $fitness_1 = f_1$
Setting 0-B: $fitness_1 = f_1\delta_1\delta_2$
Setting 0-C: $fitness_1 = f_1\delta_1\delta_2 \cdot g(f_1\delta_1\delta_2, \prod_{k=3}^{5} \delta_k,\ \rho)$

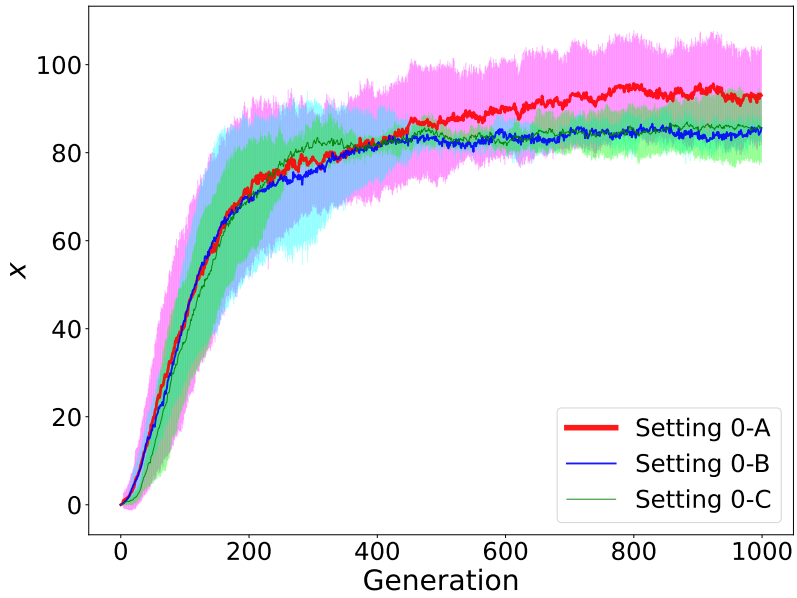


Fig. 4.4. The x-coordinate value of the final robot positions in each generation. Lines are the average over 5 trials. The error bar shows the standard deviation.

· Exp-1

Exp-1 is conducted in Env-1. The purpose of Exp-1 is to evolve a collective behavior of a multi-legged robotic swarm. The evolved controllers in Setting 0-C are set as the initial population in Exp-1 for accelerating the evolution process. Three experimental settings are conducted as follows:

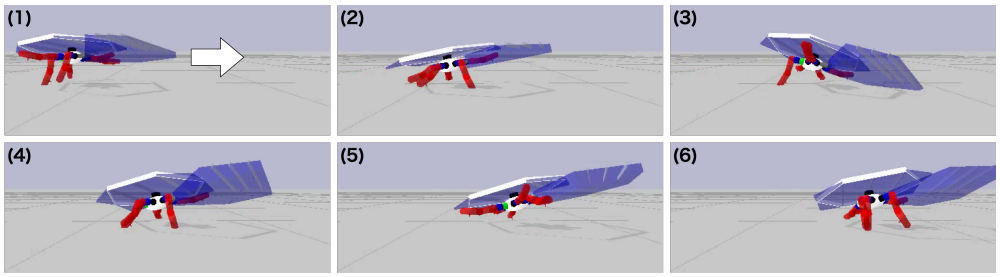
Setting 1-A: The standard setting.
Setting 1-B: The setting without using evolved controllers in Exp-0.
Setting 1-C: The setting without using $fitness_2$.

Setting 1-B is for checking the effect of the incremental evolution based on environments. Setting 1-C tests how does $fitness_2$ affects the gait of robots.

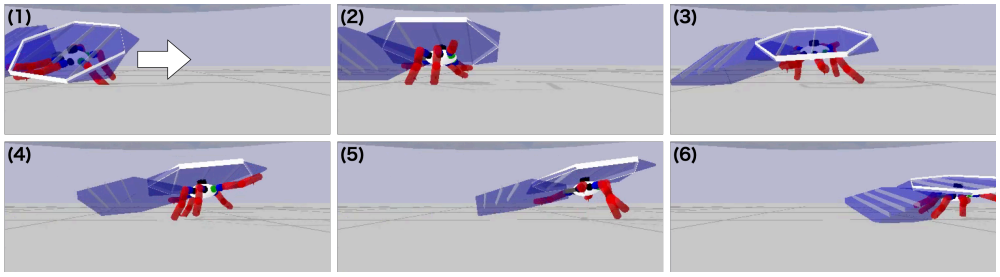
4.3 Results and Discussion

4.3.1 Evolving a Gait for a Single Robot (Exp-0)

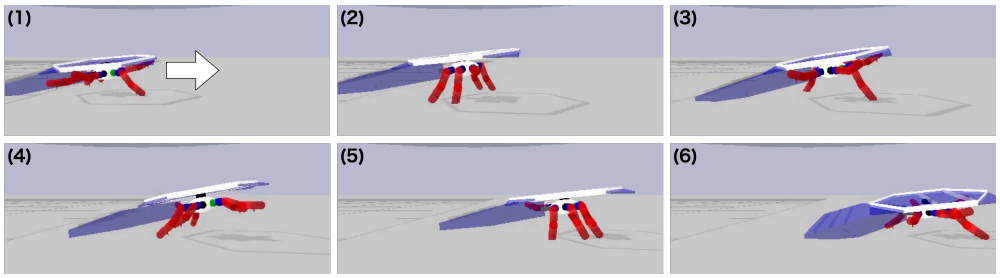
In Exp-0, different fitness functions are used in Settings 0-A, 0-B, and 0-C respectively. Therefore, instead of fitness transitions, Fig. 4.4 shows the x-coordinate values of the final robot positions in each generation. Fig. 4.4 shows that the line in Setting 0-A converged to a slightly higher value than in other settings. This means that the supplemental fitness parts slightly affect the travel distance of the robot. Fig. 4.5 shows the examples of obtained



(a) Setting 0-A.



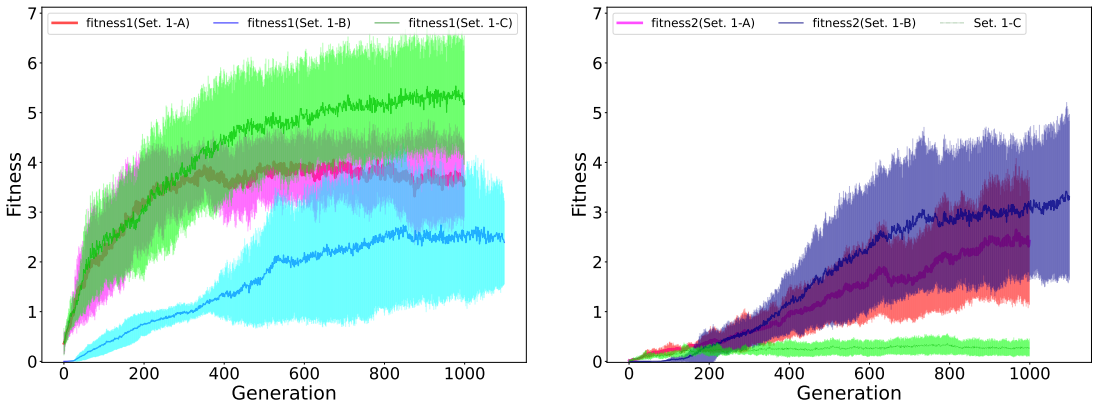
(b) Setting 0-B.



(c) Setting 0-C.

Fig. 4.5. Example of obtained behavior in Exp-0. In each setting, six pictures are taken from fixed points of view. The white arrows in the pictures show the direction the robot is waking. (a) The robot obtains backward walking through some evolution trials. (b) The robot shows forward walking while swinging the body around the roll angle. (c) The robot keeps posture while walking.

behaviors in Exp-0. Fig. 4.5(a) shows that the evolution process without δ_1 to δ_5 sometimes generates backward walking. Fig. 4.5(b) shows the example of robot behavior in Setting 0-B. By using δ_1, robots obtained forward walking in all five trials. However, the robot tends to swing the body around the roll angle while walking. Fig. 4.5(c) shows the example of robot behavior in Setting 0-C. The robot keeps the orientation and position of the body. Supplemental fitness parts successfully designed the gait similar to a natural organism.



(a) $fitness_1$ (for moving along x-axis). **(b)** $fitness_2$ (for following other robots).

Fig. 4.6. The details of the fitness transitions. Note that $fitness_2$ in Setting 1-C is not used for the final fitness calculation($K_2 = 0$). In (b), the plot corresponding to Setting 1-C is the supplemental data to compare the ability to follow other robots.

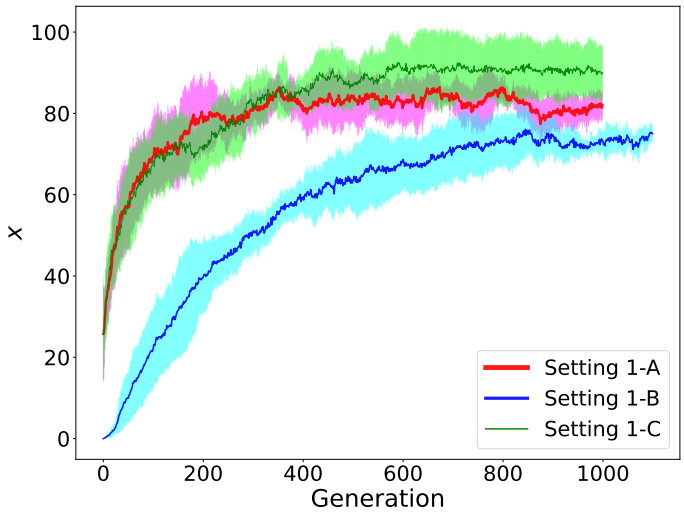
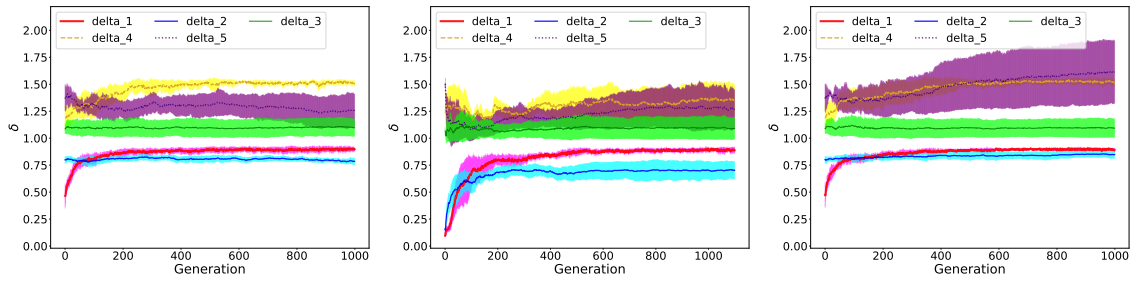


Fig. 4.7. The mean x-coordinate value of 10 robots in each generation. Lines are the average over 5 evolutionary trials.

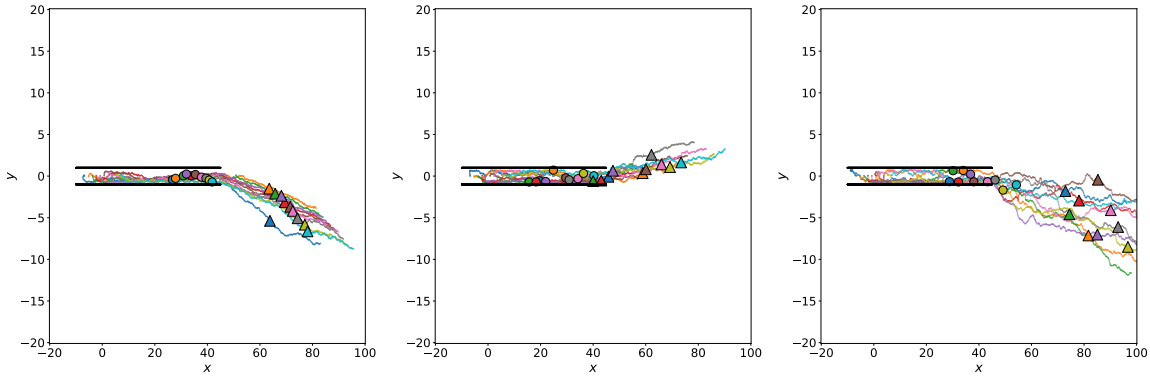
4.3.2 Evolving a Collective Behavior (Exp-1)

Fig. 4.6 shows fitness transitions in Exp-1. Settings 1-A and 1-C use controllers evolved in Setting 0-C of Exp-0 as the initial population. Therefore, Setting 1-B is conducted until 1100 generation to equalize the computational cost. In Setting 1-C, only $fitness_1$ is applied during the evolution. Fig. 4.6(a) shows that $fitness_1$ in Setting 1-C becomes the highest in all settings. In Setting 1-A, $fitness_1$ stopped increasing around 400 generations and slightly decreased. Instead of $fitness_1$, $fitness_2$ increased until the last generation, as can be seen



(a) Setting 1-A. **(b)** Setting 1-B. **(c)** Setting 1-C.

Fig. 4.8. Transitions of δs (supplemental fitness parts) in Exp-1.



(a) Setting 1-A. **(b)** Setting 1-B. **(c)** Setting 1-C.

Fig. 4.9. Robot trajectories observed in Exp-1. The circle markers show the robot positions at 2000 timestep. The triangle markers show the robot positions at 4000 timestep.

from Fig. 4.6(b). In Setting 1-A, the value of $fitness_1$ is higher than $fitness_2$. On the other hand, in Setting 1-B, the value of $fitness_2$ is slightly higher than $fitness_1$ in the latter half of the evolution. The difference between Settings 1-A and 1-B is whether or not pre-evolved controllers are used as the initial population. The results of Settings 1-A and 1-B show that the prior evolution leads to a different evolution path in Exp-1.

Fig. 4.7 shows the final x-coordinate values of robots in each generation. Fig. 4.7 shows that the setting with higher $fitness_2$ tends to show shorter traveled distances of robots. Fig. 4.8 shows the transitions of δ_1 to δ_5 in Exp-1. Fig. 4.8 shows that the values of δ_5 (prevent vibrations on joints) in Setting 1-A and 1-B become lower than Setting 1-C. Additionally, the values of δ_2 (move each leg equally) and δ_4 (keep the distance between the body and the floor) in Setting 1-B are lower than Setting 1-A. This means the setting with higher $fitness_2$ tends to show lower values in some δs. From the results of Fig. 4.6 (the higher $fitness_2$, the lower $fitness_1$), Fig. 4.7 (the higher $fitness_2$, the shorter traveled distance), and Fig. 4.8 (some δ values become low in the settings of obtaining $fitness_2$), robots seem

to change their gaits that contribute to $fitness_1$ for obtaining $fitness_2$. For following other robots (i.e., for keeping relative position and direction to a preceding robot), robots are required to obtain different behaviors from a single robot scenario, such as limiting a walking velocity, adjusting moving direction more frequently, and so on. Obtaining both $fitness_1$ and $fitness_2$ at high levels becomes a future challenge.

Fig. 4.9 shows examples of robot trajectories in each setting. These plots show the effect of $fitness_2$. In Settings 1-A and 1-B (evolved with $fitness_2$), all robots show similar trajectories, but in Setting C (evolved without $fitness_2$), the robots have spread out. By using $fitness_2$, the gaits of robots were evolved to follow other robots while robots were moving. The results of Exp-1 showed that the ER approach successfully designed how to use each joint for collective behavior of a multi-legged robotic swarm.

4.4 Conclusions

This chapter focused on the neuroevolution approach for generating collective behavior of the multi-legged robotic swarm. The recurrent neural network was employed as the robot controller. Computer simulations showed that the proposed fitness functions successfully designed the robot's gait similar to natural organisms. The results also showed that the neuroevolution approach designed the controller to generate collective behavior of the multi-legged robotic swarm.

Chapter 5

Evolving Collective Behavior in a Rough Terrain Environment

This chapter focuses on evolving collective behavior in a rough terrain environment. In Chapter 4, the neuroevolution approach successfuly designed collective behavior in flat surface. This result is extended to the advanced task: path formation task in rough terrains with 20 robots. The multi-legged robots show three-dimensional behaviors, such as climbing steps or obstacles by coordinating thier legs. Therefore, the multi-legged robotic swarm is expected to exhibit collective behaviors in rough terrains. However, in these task scenarios, the robot controller has to adapt to height difference on the floor beside the original task requirement. This implies the controller design becomes more challenging than the flat surface. In this chapter, incremental evolution with environmental transitions was employed to achieve the task in the rough terrain. The similar approach to Chapter 4 was employed to evolve a robotic swarm. The results of computer simulations showed that the neuroevolution is also promising to the rough terrain scenario.

This chapter is organized as follows. Section 5.1 describes experimental settings. Section 5.2 describes the details of the neuroevolution approach. Section 5.3 shows the experimental results. Section 5.4 discusses the adaptability of obtained controllers. Finally, Section 5.5 present the conclusion.

5.1 Settings of Experiments

This section describes the task settings including environment and robot specifications. The experiment of this chapter aims to generate collective behavior of a multi-legged robotic swarm in a rough terrain environment. The robot controller is obtained by a neuroevolution approach. The performance of a robotic swarm is evaluated in a path formation task. The experiment is conducted by computer simulations as in the former chapters.

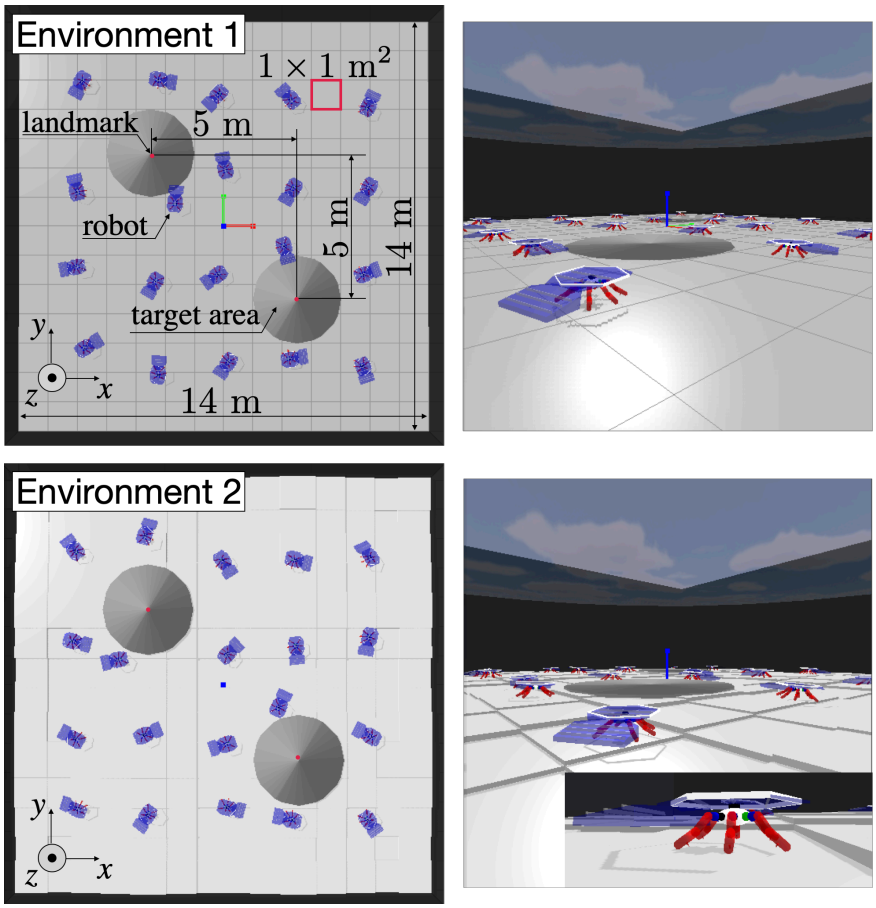


Fig. 5.1. Environmental settings for the path formation task. Environment 1 (Env-1) is a square arena with a flat field. Environment 2 (Env-2) has the rough terrain field that consists of cuboid blocks.

5.1.1 Environmental settings

Fig. 5.1 shows the environmental settings of the path formation task. The evolution processes are executed in two environments, i.e., *environment 1* (Env-1) and *environment 2* (Env-2). Env-1 and Env-2 are square arenas that include two target areas. In the task, robots are required to visit two target areas alternately. The landmark is placed at the center of a target area. Landmarks are equipped with a colored LED that can be detected by a robot. In Env-1, the floor is a flat field, while Env-2 has a rough terrain field that consists of cuboid blocks. Fig. 5.2 shows the settings of the block arrangement in Env-2. Blocks are positioned to shape a sine wave surface. The difficulty of moving in the rough terrain field is determined by Δh and ω in Fig. 5.2. In both environments, a total of 20 robots are placed with random initial facing directions. The task period is set as 5000 timesteps. The robot controller gets sensor inputs and decides the output at each timestep.

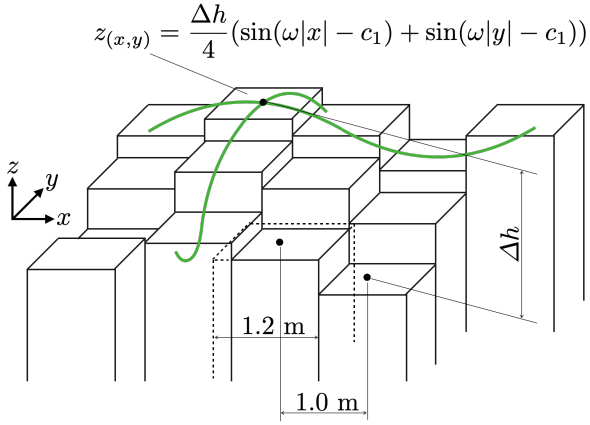


Fig. 5.2. The setting for floor blocks in Env-2. Blocks are arranged to shape a sine wave surface. The Δh shows the height difference between the highest point and the lowest point on the surface. The ω is the frequency of a sine wave.

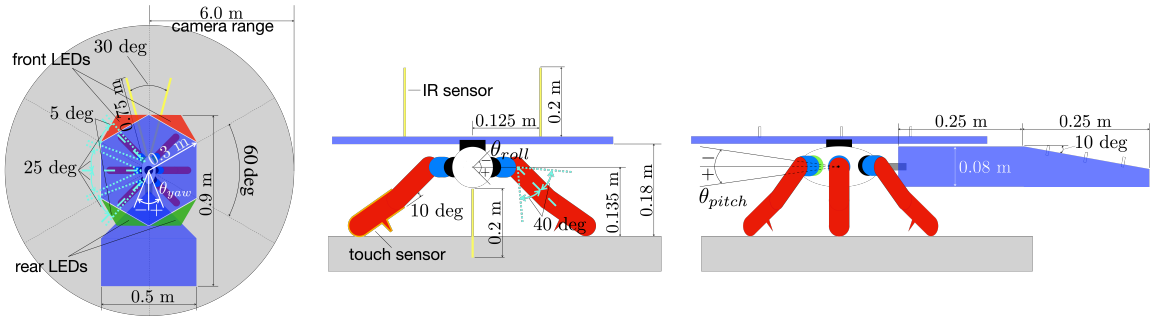


Fig. 5.3. The robot specifications. The main change from Chapter 4 is the visible range of the camera; the robot has an omnidirectional camera with a longer sight range. In addition, the bottom IR sensor obtains the new ability to detect the target areas. On the other hand, robots lost an electric compass. These settings are employed to discuss a path formation task.

5.1.2 Robot settings

Fig. 5.3 illustrates the robot specifications. The body structure of the robot is similar to Chapter 4, while sensor settings are slightly changed. The first change is the visible range of the camera; the omnidirectional camera is employed. As in former chapters, the camera range has six sections. Each section detects the colored LEDs on the front and back of the robot independently. In total, twelve binary signals are obtained from the camera. The second change is in the bottom IR sensor; it also acts as the "ground sensor". The ground sensor returns 1 when detects the target area and 0 otherwise. Finally, the electric compass is removed because the compass makes the task significantly easy.

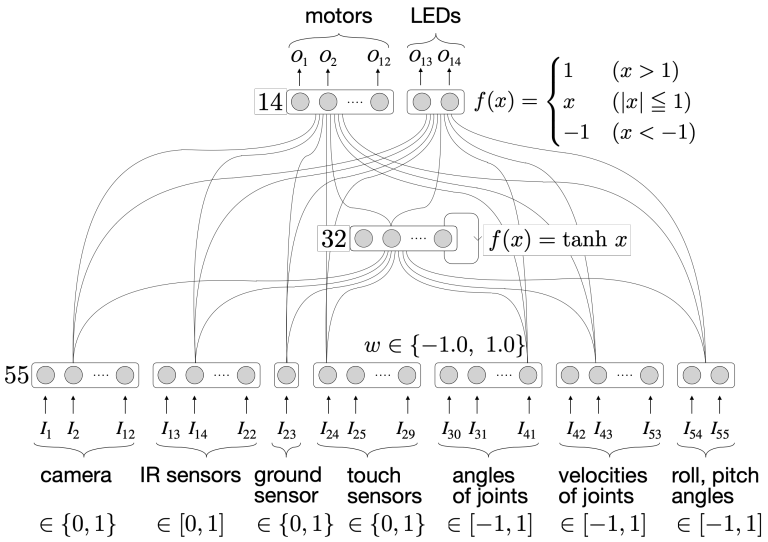


Fig. 5.4. Structure of the robot controller.

5.2 Methods

The neuroevolution approach is employed to design a robot controller. This section describes the method to apply neuroevolution for the path formation task of the multi-legged robotic swarm.

5.2.1 Controller

Fig. 5.4 illustrates the structure of the robot controller. The controller structure is similar to Chapter 4 except for the number of neurons on input and hidden layers. The input layer consists of 55 neurons which obtain inputs from a camera, IR sensors, a ground sensor, touch sensors, angles and velocities of joints, a roll angle of the torso, and a pitch angle of the torso. The output layer has 14 neurons. Twelve of them decide to target angular velocities of joints. The remaining two turn on LEDs if the output values are larger than 0.5. A total of 4002 synaptic weights are optimized. This study also employs (μ, λ)-ES as in Chapter 4. Table 5.1 summarizes the parameter settings of the (μ, λ)-ES. The main change from Chapter 4 is tripled population size; $\lambda = 192$ is determined by preliminary experiments for more difficult tasks than Chapter 4.

5.2.2 Fitness Function

The performance of a robotic swarm on the task is evaluated by the fitness function. The fitness function consists of two parts; $fitness_1$ is for walking around in an environment, and $fitness_2$ is for visiting target areas. The $fitness_1$ has a similar structure to Chapter 4,

Table 5.1. Parameter settings of the (μ, λ)-ES.

Parameter	Value
Number of weights n	4002
Number of parents μ	192
Number of offspring λ	64
Mutation step size $\boldsymbol{\sigma}$	$\in [0.0005, 0.15]$
Initial mutation step size	$\sim U(0.004, 0.006)$
Terminate generation $G_{\max}$	1000
Proportional constants (c, c')	(0.5, 2.5)

which is represented as follows:

$$fitness_1 = f_v \cdot \delta_1\delta_2 \cdot g(f_v\delta_1\delta_2, \delta_3\delta_4\delta_5). \tag{5.1}$$

$$f_v = \min\left(a\bar{v}, a(2v_{\max} - \bar{v}), v_{\max}\right)$$

$$\bar{v} = \frac{1}{N_r}\sum_{i}^{N_r} ||\boldsymbol{v}_i||_2 \tag{5.2}$$

where a, N_r, and $v_{\max}$ are constant values. The N_r is the number of robots. The $v_{\max}$ is the target speed of walking robots. The $\boldsymbol{v}_i$ is the velocity vector of a robot i. The f_v has a trapezoid shape that is centered at $v_{\max}$. The f_v requires robots to move at the approximate speed $v_{\max}$. In Eq. 5.1, δ_1 to δ_5 are supplemental parts to make a robot's gait similar to natural organisms. The representations of δ_1 to δ_5 are the same to Chapter 4. The roles of δ_1 to δ_5 are as follows: δ_1 is for walking in the front direction of the robot, δ_2 is for using each leg equally, δ_3 is for keeping roll and pitch angles of the body, δ_4 is for keeping the distance between the body and the floor, and δ_5 is for preventing vibrations on joints. Finally, the function $g(x, y)$ in Eq. 5.1 is given as follows:

$$g(x, y) = \begin{cases} y & x \geq \alpha \\ 1 & \text{otherwise} \end{cases} \tag{5.3}$$

where α is the constant value. The $g(x, y)$ realizes a hierarchical training by using α as a threshold. The δ_3, δ_4, and δ_5 become available after f_v, δ_1, and δ_2 are trained to some extent. This setting is also similar to Chapter 4. Consequently, $fitness_1$ requires robots to walk around the environment.

The $fitness_2$ (for visiting target areas) is represented as follows:

$$fitness_2 = \sum_{i=1}^{N_r} f_{\text{target},i} \tag{5.4}$$

$$f_{\text{target},i}(t) = f_{\text{target},i}(t-1) + \begin{cases} 1 & \text{if robot } i \text{ enters a target area,} \\ 0 & \text{otherwise} \end{cases}$$

where t shows the timestep. The $fitness_2$ requires robots to visit two target areas alternately. At last, the total fitness is represented as follows:

$$F = K_1 \cdot fitness_1 + K_2 \cdot fitness_2 \tag{5.5}$$

where K_1 and K_2 are constant values.

5.2.3 Incremental Evolution

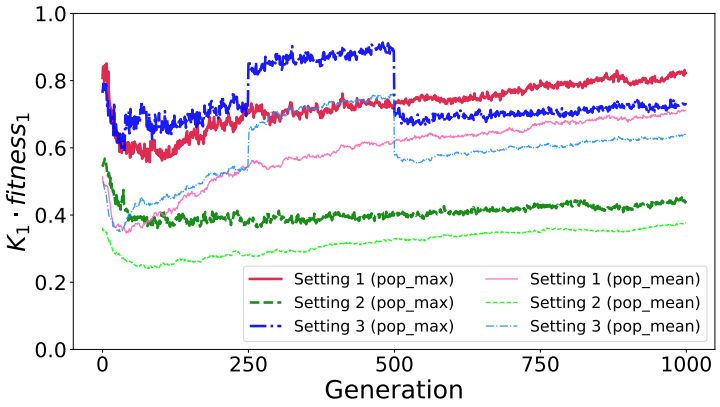
Incremental evolution is a promising method to generate complex behavior in robots [104]. This study employs incremental approaches based on staged evolution and environmental complexification [104]. Before the path formation task, preliminary evolution is conducted using a single robot with $fitness_1$ (i.e., $K_2 = 0$) in Env-1. The preliminary evolution aims to make a robot walk in a flat field. The robot controllers that pass the preliminary evolution are set as the initial population $\boldsymbol{\mathcal{P}}_G$ of the path formation task. The three evolution settings are applied for the $\boldsymbol{\mathcal{P}}_G$ as follows:

Setting 1: Evolve for 1000 generations in Env-1.
Setting 2: Evolve for 1000 generations in Env-2 ($(\Delta h, \omega) = (0.2, 1.0)$).
Setting 3: Evolve for 250 generations in Env-1, and use Env-2 (250 generations for $(\Delta h, \omega) = (0.1, 1.0)$ and 500 generations for $(\Delta h, \omega) = (0.2, 1.0)$).

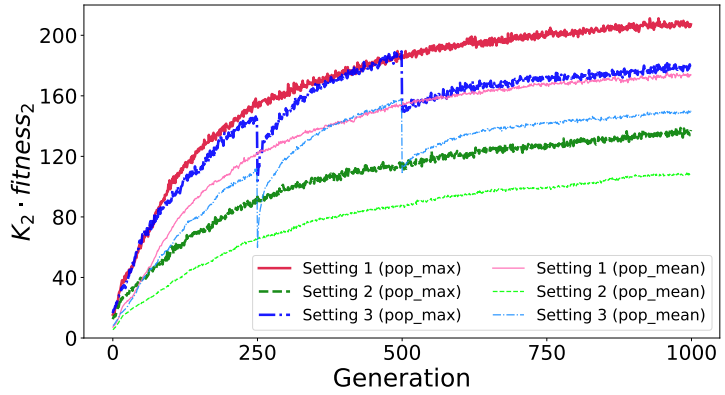
Setting 1 is for achieving the path formation task in the flat field. Setting 2 and Setting 3 are for achieving the task in the rough terrain field. Setting 2 uses a direct evolution in Env-2 with the terrain setting $(\Delta h, \omega) = (0.2, 1.0)$. Setting 3 uses an incremental evolution based on environmental transitions. In the evolution process, $fitness_1$ becomes a supplemental part (i.e., $K_2 \cdot fitness_2 \gg K_1 \cdot fitness_1$).

5.3 Results

In the experiment, five evolution trials are conducted for Setting 1 to Setting 3. The fitness transitions are shown in Fig. 5.5. Lines plotted the max and mean fitness of a population in each generation. Each line is averaged over five evolution trials. In Fig. 5.5(a), the value of $fitness_1$ decreased in the initial phase of evolution. After that, the value of $fitness_1$ slightly increased. This phenomenon could be caused by the prioritization of $fitness_2$ over $fitness_1$. Fig. 5.5(a) also shows that the $fitness_1$ of Setting 3 increased after using Env-2 from the 250 generations ($(\Delta h, \omega) = (0.1, 1.0)$). However, the value decreased again after

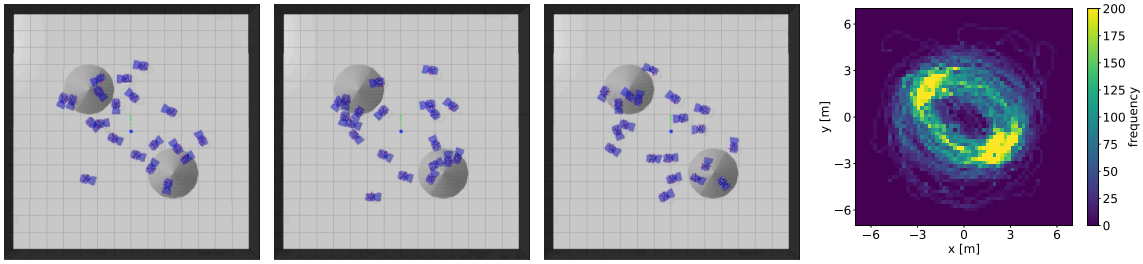


(a) $K_1 \cdot fitness_1$.



(b) $K_2 \cdot fitness_2$.

Fig. 5.5. Fitness transitions. Each plot is averaged over five evolution trials.

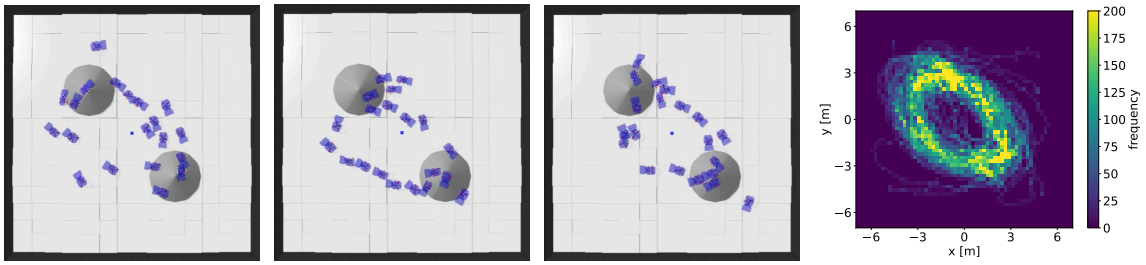


(a) 2000 step. **(b)** 3000 step. **(c)** 4000 step. **(d)** heatmap.

Fig. 5.6. Example of observed behavior in Setting 1.

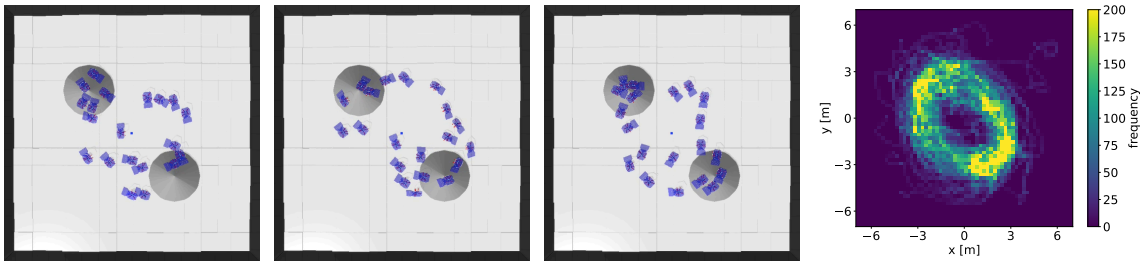
the 500 generations ($(\Delta h, \omega) = (0.2, 1.0)$). These results indicate the field with small steps is easier for robots to move than the fully flat field.

In the experiment, the value of K_2 is set as 1.0. Therefore, $K_2 \cdot fitness_2$ in Fig. 5.5(b) shows the number of arrivals to target areas. In Setting 1, $fitness_2$ increased gradually and converged around 1000 generations. The best controllers of Setting 1 recorded about 200 times of arrivals. In Setting 2, the $fitness_2$ also converged around 1000 generations,



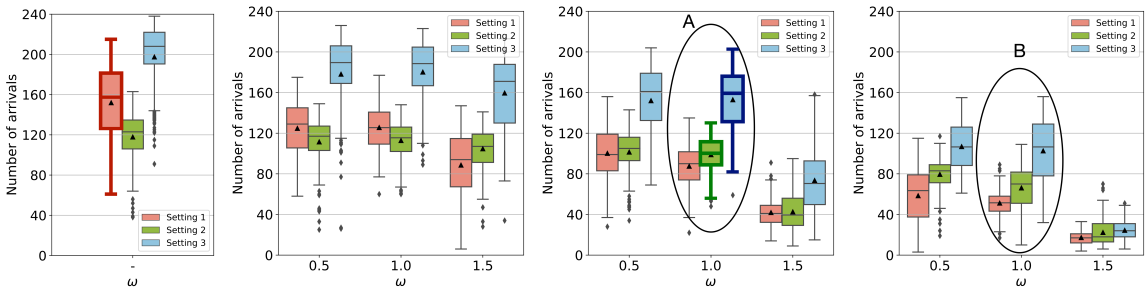
(a) 2000 step. **(b)** 3000 step. **(c)** 4000 step. **(d)** heatmap.

Fig. 5.7. Example of observed behavior in Setting 2.



(a) 2000 step. **(b)** 3000 step. **(c)** 4000 step. **(d)** heatmap.

Fig. 5.8. Example of observed behavior in Setting 3.

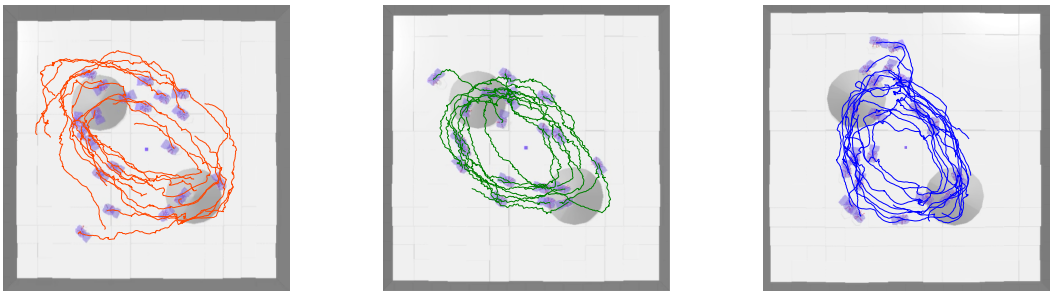


(a) $\Delta h = 0.0$. **(b)** $\Delta h = 0.1$. **(c)** $\Delta h = 0.2$. **(d)** $\Delta h = 0.3$.

Fig. 5.9. Boxplots for flexibility tests. The boxes with bold lines show the terrain setting where controllers are obtained.

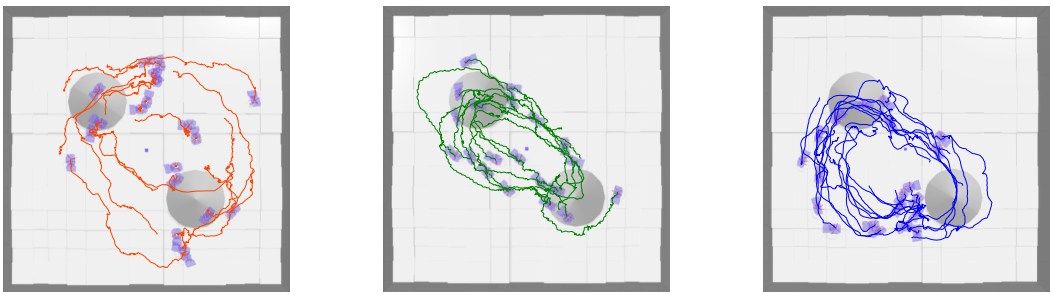
while the fitness value is lower than in Setting 1. The best controllers of Setting 2 recorded about 140 times of arrivals. The curve of Setting 3 shows a sharp decline corresponding to the environmental transitions. However, in the last generation, Setting 3 shows better performance than Setting 2 despite the same terrain setting. This result showed that incremental evolution is effective for a multi-legged robotic swarm in rough terrain fields.

Fig. 5.6 to Fig. 5.8 show the observed behaviors of the robotic swarm in each setting. Add to snapshots, the long-term records about robot positions are summarized as two-dimensional histograms. Fig. 5.6 shows that the multi-legged robotic swarm successfully forms a path



(a) Setting 1. **(b)** Setting 2. **(c)** Setting 3.

Fig. 5.10. Observed behavior in $(\Delta h, \omega) = (0.2, 1.0)$ ("A" in Fig 5.9(c)).



(a) Setting 1. **(b)** Setting 2. **(c)** Setting 3.

Fig. 5.11. Observed behavior in $(\Delta h, \omega) = (0.3, 1.0)$ ("B" in Fig 5.9(d)).

formation in the flat field. Fig. 5.7 and Fig. 5.8 show that path formations are also achieved in the rough terrain field. These results showed that the neuroevolution approach successfully designed path formation behavior in the flat field and the rough terrain field.

5.4 Discussion

This section discusses the flexibility of the robotic swarm for terrain variations. The robot controllers are applied for terrain settings where controllers are not evolved. The Δh in Env-2 is varied as $\{0.0, 0.1, 0.2, 0.3\}$. The ω is also varied as $\{0.5, 1.0, 1.5\}$. A total of 10 terrain settings (ω is irrelevant when $\Delta h = 0.0$) are employed. The robot controller with the best performance is selected from each evolution trial. The five controllers from each Setting 1, 2, and 3 are applied to 10 terrain settings. Each controller is evaluated 30 times.

The results are summarized in Fig. 5.9. Each box consists of 150 scores (30 plots $\times$ 5 controllers). Fig. 5.9 shows that the performance of the robotic swarm decreased as Δh increased. In addition, changes of ω from 0.5 to 1.0 are not critical to the performance, while 1.0 to 1.5 seriously affect the performance. In terrain settings except $(\Delta h, \omega) = (0.2, 0.5)$ and $(0.2, 1.5)$, controllers of Setting 1 and Setting 2 showed significant differences among performances (Kruskal-Wallis test, $p_{KW} < 0.05$ and Bonferroni-corrected Mann-Whitney U

test, $p_U < 0.05/3$). When the terrain setting is close to the flat field ($\Delta h = 0$, $(\Delta h, \omega) = (0.1, 0.5)$, $(0.1, 1.0)$), controllers of Setting 1 show better performances, whereas controllers of Setting 2 are better in rougher terrain settings. Additionally, in all terrain settings, controllers of Setting 3 showed better performances than Setting 1 and 2($p_{KW} < 0.05$, $p_U < 0.05/3$). This shows the incremental evolution approach designs controllers which can cope with a wide range of terrain settings.

Fig. 5.10 shows the example of observed behavior in the terrain setting of $(\Delta h, \omega) = (0.2, 1.0)$ ("A" in Fig 5.9(c)). The background pictures show the robot's positions at the final timestep of the task. The robot trajectories are plotted for the last 500 timesteps. Fig. 5.10 shows that the controller of Setting 1 could generate a path formation. This means the controllers evolved in a flat field can cope with a rough terrain field. Additionally, Fig. 5.11 shows the example of observed behavior in $(\Delta h, \omega) = (0.3, 1.0)$ ("B" in Fig 5.9(d)). In this field, controllers from Setting 2 and Setting 3 outperform the controllers of Setting 1. In Fig. 5.11, some robots of Setting 1 could not move in the environment sufficiently while robots of Setting 2 and Setting 3 keep a path formation. The controllers evolved in a rough terrain field show better flexibility for more difficult terrain settings.

5.5 Conclusions

This chapter focused on evolving collective behavior in a rough terrain environment. The neuroevolution approach proposed in Chapter 4 was extended to a path formation task with 20 robots. The experimental results showed that the neuroevolution succeeded in generating collective behavior not only on a flat but also on rough terrains. The results also showed that incremental evolution is an effective approach for a multi-legged robotic swarm in rough terrain scenarios.

Chapter 6

Generating and Analyzing Collective Step-Climbing Behavior

This chapter presents the neuroevolution approach for generating and analyzing collective step-climbing behavior. The method in Chapter 4 retries to achieve the step-climbing task. In addition, this chapter aims to analyze what kinds of behaviors are obtained through the evolution process. The task setting seems to evolve an ability to climb the step or other robots. On the other hand, preliminary experiments imply that robots show several altruistic behaviors such as keeping postures in front of the step, and making wide stepping stones; these behavior seem to contribute to achieving the task. To examine the obtained behavior of robots, four measurement factors are proposed. The evolution process is evaluated by the performance of robots and the values of measurement factors. The experimental results showed that the transitions of measurement factors support the hypothesis about obtained behaviors.

The rest of this chapter is organized as follows. Section 6.1 describes experimental settings on computer simulation. Section 6.2 describes the details of measurement factors. Section 6.3 shows the experimental results and discussion. Finally, Section 6.4 conclude this chapter.

6.1 Settings of Experiments

This experiment aims to generate collective step-climbing behavior of a multi-legged robotic swarm by using the neuroevolution approach. The experiment also focuses on analyzing the obtained behavior. Fig. 6.1 shows the environmental settings of the experiment. In this study, the evolution process starts from the situation where the controller can make robots walk and form a line; this setting focuses on evolving step-climbing behavior. The preliminary evolution for obtaining walking and forming a line was conducted as in Chapter 4. The target step is higher than a single robot. Therefore, robots have to utilize other robots

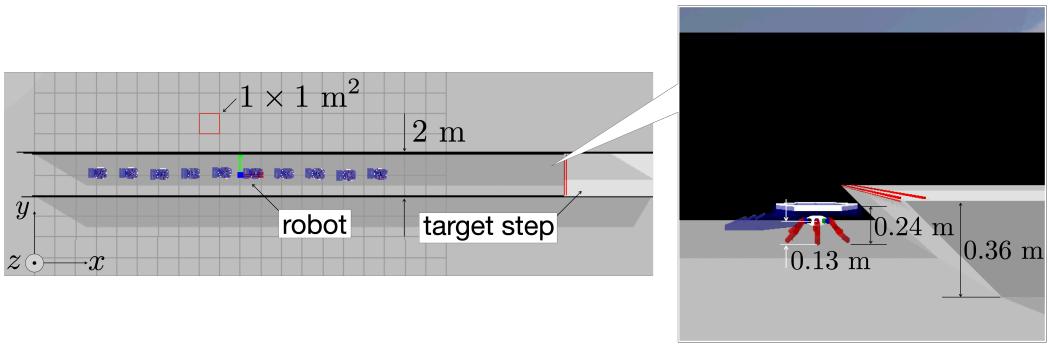


Fig. 6.1. The experimental environment. In this study, the evolution process starts from the situation where the robots can walk along the x-axis and form a line. The preliminary evolution for obtaining walking and forming a line was conducted as in Chapter 4.

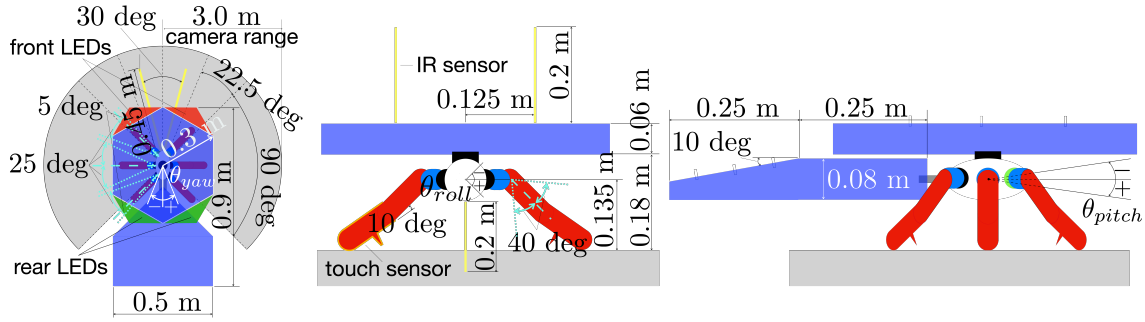
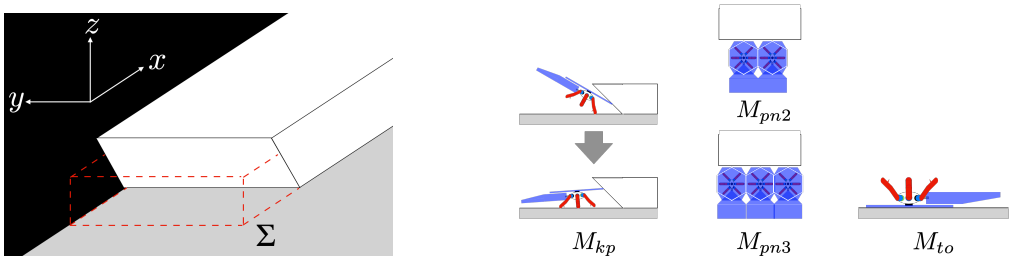


Fig. 6.2. Settings of the robot. Cyan dotted lines indicate the movable ranges of the joints. The gray circular sector shows the visible range of the camera.

as stepping-stones for achieving the task. The target step has an acute angle on the surface. The robots should keep their posture for becoming a stepping stone. If the robot climbs the step, it will be replaced at 5 m in front of the step and join the task again.

Fig. 6.2 illustrates the robot configuration. The robot specifications are similar to Chapter 4 except for the thickness of the shell. Robots use a thicker shell than Chapter 4 for more stable simulation. Additionally, the controller setting and evolutionary algorithm setting are also similar to Chapter 4. However, the population size is changed. As in Chapter 5, the numbers of parents(μ) and offspring(λ) are set as 64 and 192 for the difficulty of the task. In this experiment, the fitness function consists of three parts; $fitness_1$ for walking, $fitness_2$ for following the other robots, and $fitness_3$ for climbing the step. Fitness settings are designed based on Chapter 3 and Chapter 4. This experiment mainly uses $fitness_3$ for evolving step-climbing behavior (i.e., $K_3 \cdot fitness_3 \gg K_1 \cdot fitness_1, K_2 \cdot fitness_2$).



(a) The cuboid space specified by Σ. **(b)** Measurement factors for robot states.

Fig. 6.3. Illustrations about the measurement factors.

6.2 Measurement Factors

This study also focuses on analyzing obtained behavior of robots. The action of robots in front of the step is important to understand the physical interactions between robots. Therefore, Σ is introduced to describe the position and orientation of robots in front of the step, which is defined as follows:

$$\Sigma = \begin{cases} 1 & \text{if} \quad x_\Sigma < x \quad \text{and} \quad z < z_\Sigma \quad \text{and} \\ & \quad |\theta_{roll}| < \theta_{r\_\Sigma} \quad \text{and} \quad |\theta_{pitch}| < \theta_{p\_\Sigma} \quad \text{and} \quad |\theta_{yaw}| < \theta_{y\_\Sigma} \\ 0 & \text{otherwise} \end{cases} \tag{6.1}$$

where x and z are the coordinate values of the robot. The θ_{roll}, θ_{pitch}, and θ_{yaw} are the orientation angles of the robot. The x_Σ, z_Σ, θ_{r_Σ}, θ_{p_Σ}, and θ_{y_Σ} are threshold values. The Σ judges whether or not the robot is standing in front of the step and facing toward the step. Fig. 6.3(a) shows the cuboid space specified by Σ. The thresholds are set as follows; $x_\Sigma = 15.2$[m], $z_\Sigma = 0.19$[m], $\theta_{r\_\Sigma} = 40$[deg], $\theta_{p\_\Sigma} = 30$[deg], $\theta_{y\_\Sigma} = 60$[deg].

Four measurement factors are introduced for detecting the situations of robots. Fig. 6.3(b) illustrated the robot's situations corresponding to measurement factors. These situations are determined by the observation of robots in preliminary experiments. The M_{kp} is the measurement of keeping the posture of the robot, which is calculated by the following equations:

$$M_{kp} = \frac{1}{N_r} \sum_{t}^{T} \sum_{i}^{N_r} f_{kp,t,i} \tag{6.2}$$

$$f_{kp,t,i} = \begin{cases} \theta_{pitch,i} & \text{if} \quad \text{the robot } i \text{ satisfies } \Sigma = 1 \text{ at the timestep } t \\ 0 & \text{otherwise} \end{cases}$$

where $\theta_{pitch,i}$ is the pitch angle of the ith robot. M_{kp} is the average pitch angle among robots that satisfy $\Sigma = 1$. If the robots keep their posture, M_{kp} will become negative or close to zero based on the pitch angle defined in Fig. 6.2.

The M_{pn2} and M_{pn3} are measurements for the spatial arrangement of robots in front of

the step. M_{pn2} is calculated by the following equations:

$$M_{pn2} = \sum_{t}^{T} f_{pn2,t} \tag{6.3}$$

$$f_{pn2,t} = \begin{cases} 1 & \text{if} \quad \text{robot } i,\ j(i \neq j) \text{ satisfy } \Sigma = 1 \quad \text{and} \\ & \quad \theta_{pitch,i},\ \theta_{pitch,j} \in (\underline{\theta}_{p\_pn},\ \overline{\theta}_{p\_pn}) \quad \text{and} \\ & \quad ||\boldsymbol{x}_i - \boldsymbol{x}_j||_2 < r_{pn2} \\ 0 & \text{otherwise} \end{cases}$$

where $\underline{\theta}_{p\_pn}$ and $\overline{\theta}_{p\_pn}$ are thresholds about the pitch angle. The r_{pn2} is the threshold for the distance between robot i and j. The absolute values of $\underline{\theta}_{p\_pn}$ and $\overline{\theta}_{p\_pn}$ are set to smaller values than the θ_{p_Σ}. Therefore, M_{pn2} sets a more narrow range about the pitch angle than Σ. The M_{pn2} detects the situation where two robots are positioned next to each other, as illustrated at M_{pn2} of Fig. 6.3(b). The M_{pn2} is calculated when just two robots satisfy $\Sigma = 1$. Each thresholds are set as follows; $\underline{\theta}_{p\_pn} = -45$[deg], $\overline{\theta}_{p\_pn} = 15$[deg], $r_{pn2} = 0.8$[m].

The M_{pn3} detects the situation where three robots are positioned next to each other, as illustrated in Fig. 6.3(b). M_{pn3} is calculated by the following equations:

$$M_{pn3} = \sum_{t}^{T} f_{pn3,t} \tag{6.4}$$

$$f_{pn3,t} = \begin{cases} 1 & \text{if} \quad \text{robot } i,\ j,\ k(i \neq j \neq k) \text{ satisfy } \Sigma = 1 \quad \text{and} \\ & \quad \theta_{pitch,i},\ \theta_{pitch,j},\ \theta_{pitch,k} \in (\underline{\theta}_{p\_pn},\ \overline{\theta}_{p\_pn}) \\ 0 & \text{otherwise.} \end{cases}$$

The values of $\underline{\theta}_{p\_pn}$ and $\overline{\theta}_{p\_pn}$ are the same as M_{pn2}. The M_{pn3} is calculated when just three robots satisfy $\Sigma = 1$. The M_{pn2} and M_{pn3} show the total timesteps that robots are placed like M_{pn2} and M_{pn3} in Fig. 6.3(b).

The M_{to} detects the turnovered robots, which is calculated by the following:

$$M_{to} = \frac{1}{N_r} \sum_{t}^{T} \sum_{i}^{N_r} f_{to,t,i} \tag{6.5}$$

$$f_{to,t,i} = \begin{cases} 1 & \text{if} \quad \theta_{r\_to} < |\theta_{roll,i}| \\ 0 & \text{otherwise} \end{cases}$$

where $\theta_{roll,i}$ is the roll angle of the ith robot. The θ_{r_to} is the threshold of the turnover. θ_{r_to} is set to 135[deg]. The M_{to} is calculated regardless of the Σ.

6.3 Results and Discussion

In this study, a total of ten evolutionary processes are conducted. Fig. 6.4 shows the number of robots that have climbed the step in each generation. In addition, Fig. 6.5 and Fig. 6.6 show observed behaviors in the initial and last generations, respectively. In Fig. 6.5, robots could not achieve the task; some of the robots did not keep postures in front of the

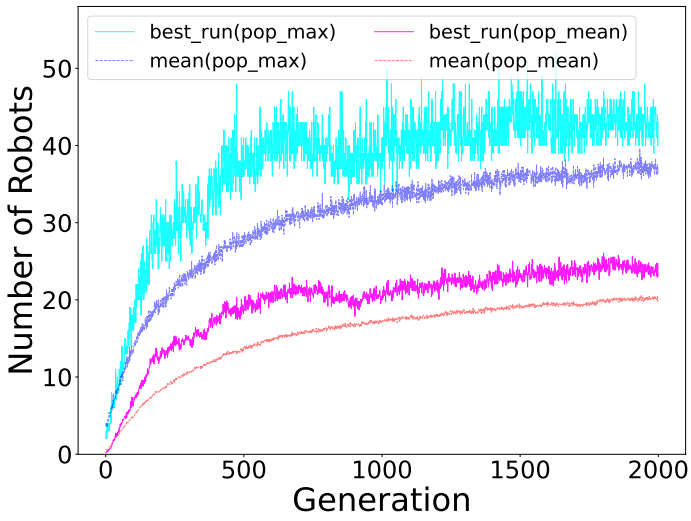


Fig. 6.4. The number of robots that have climbed the step. The dashed lines are mean values over ten evolution trials. The solid lines show the best run.

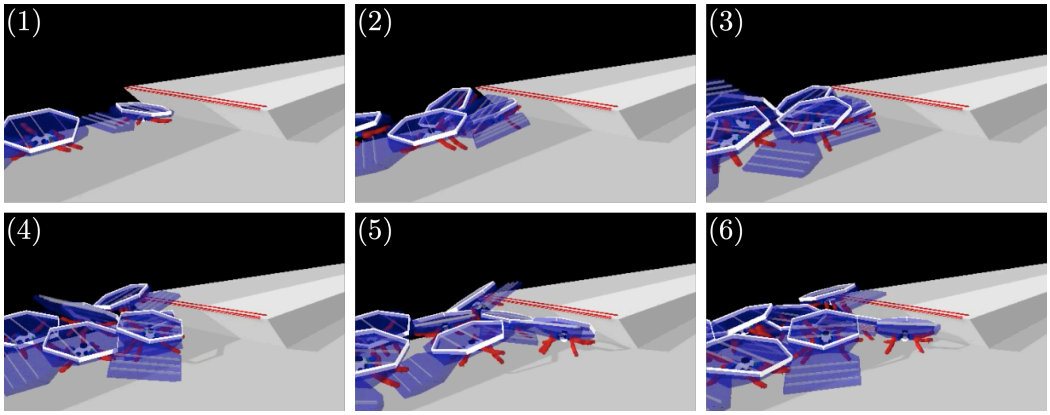


Fig. 6.5. Observed behavior in the initial generation.

step. On the other hand, the robots in Fig. 6.6 succeeded in achieving the task by building stable stepping stones. These results showed that the neuroevolution approach succeeded in designing the robot controllers for achieving the task.

Additionally, Fig. 6.7 shows the transitions of measurement factors. In Fig. 6.7, the mean value of M_{kp} decreased. This means the robots stopped tilting forward at the front of the step and kept their posture flat or behaved like a slope. This behavior seems to be better for achieving the task because the robot tilting forward becomes an additional obstacle. In addition, the figures show that the value of M_{pn3} becomes higher than M_{pn2}. This shows that the situation of M_{pn3} occurred more frequently in evolutionary processes. In Fig. 6.7, the value of M_{to} increased steeply in the initial generations and slightly decreased after around 500 generations. In the initial generation, the robots cannot climb the step. Therefore, robots

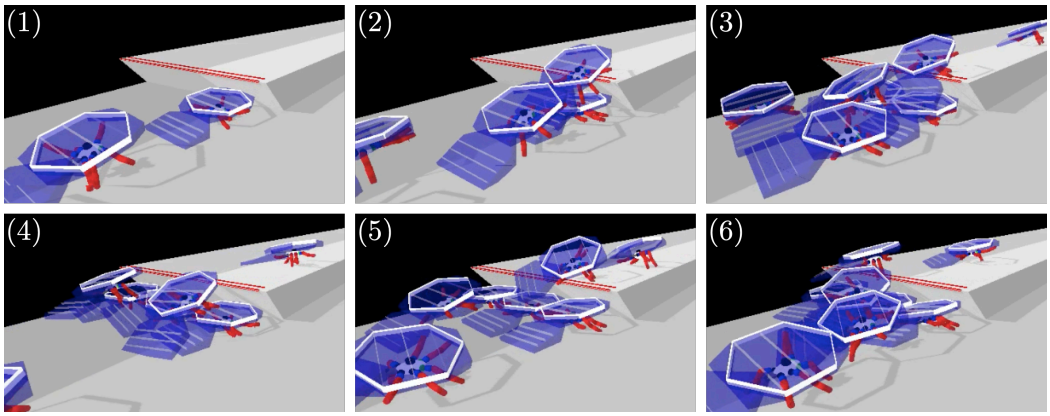


Fig. 6.6. Observed behavior in the last generation.

do not turn over frequently. Through the evolution process, robots tried to climb other robots or the step, and the risk of turnover also has increased. Subsequently, the turnovers are decreased by the selection pressure. The result of measurement factors shows that the robot obtained behavior to support other robots along with behavior to climb objects.

For further understanding of the behavior, evolved controllers are re-evaluated. The controllers with the best 10 fitnesses in each evolutionary process are selected. A total of 100 controllers are tested for 100 trials. The correlation coefficients between measurement factors and the performance of the task are summarized in Fig. 6.8. The result shows that the correlation between M_{pn2} and the performance is relatively weaker than other measurement factors. Fig. 6.7 shows that the situation of M_{pn3} occurs more frequently than M_{pn2}. Therefore, M_{pn2} seems to have little effect on the performance. The remaining factors(M_{kp}, M_{pn3}, and M_{to}) indicate a correlation coefficient of approximately 0.4 with a positive or negative sign. Fig. 6.8 shows that the robot behaviors corresponding to M_{kp}, M_{pn3}, and M_{to} contribute to achieving the task. However, correlations are not so strong.

6.4 Conclusions

This chapter focused on generating and analyzing a collective step-climbing behavior. The neuroevolution approach was applied to designing a robot controller. In addition, four measurement factors were proposed to analyze what kind of behaviors were obtained. The results of computer simulations showed that the neuroevolution successfully generated collective step-climbing behavior as in Chapter 3. This result showed the great potential of neuroevolution for designing complicated control software. In addition, the transitions of measurement factors indicated that the robots evolved to show behaviors that support other robots and contribute to achieving the task.

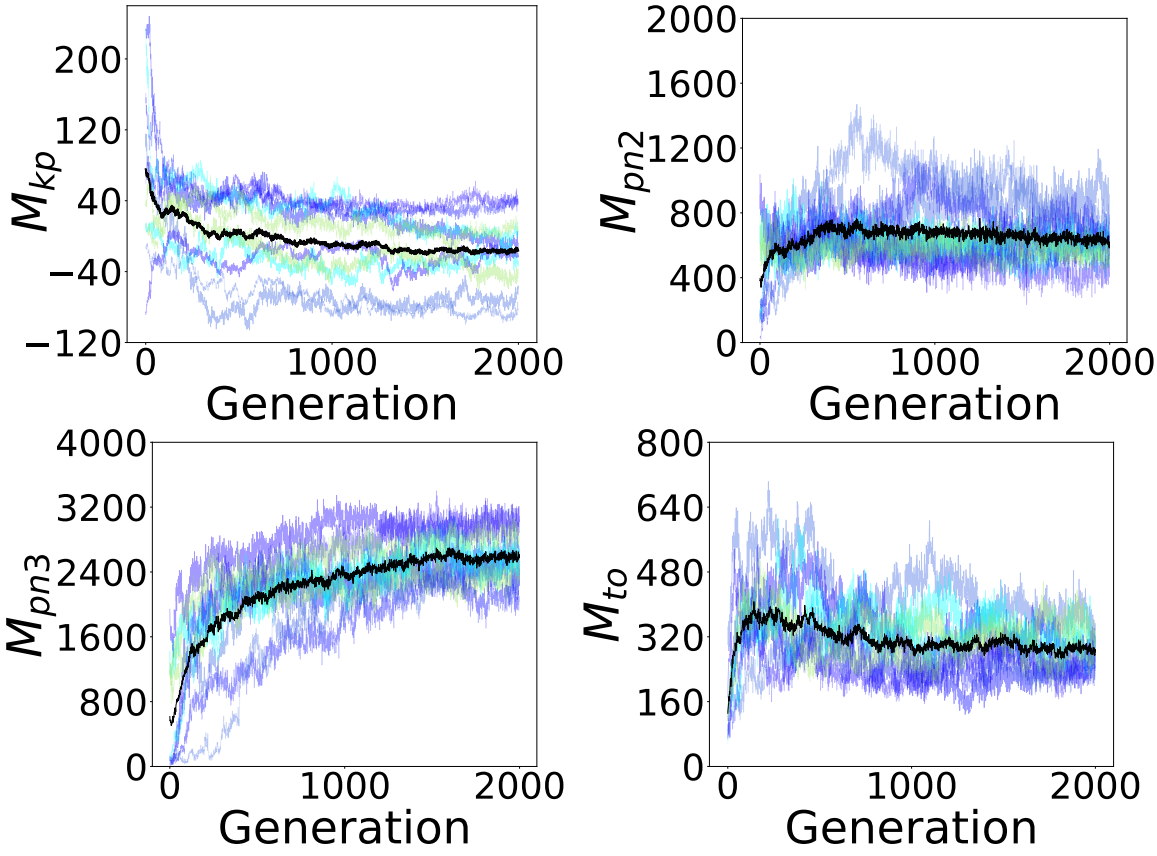


Fig. 6.7. The transitions of measurement factors. All measurement factors are calculated as the mean value of the population. Dashed lines are the results of each trial. The black solid line is the mean value over ten evolutionary trials.

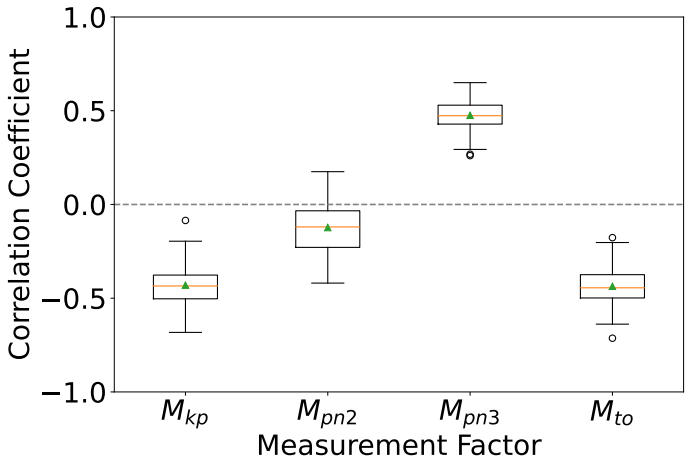


Fig. 6.8. The box plots of the correlation coefficients. Each box consists of 100 correlation coefficients. Each coefficient is calculated by 100 scores of each controller.

Chapter 7

Deep Reinforcement Learning Approach for a Multi-Legged Robotic Swarm

This chapter presents an approach based on deep reinforcement learning. In Chapters 3 to 6, all of the experiments were conducted by evolutionary robotics approaches. Evolutionary robotics is a common approach in automatic design methods due to the learnability for dynamical environmental settings; the evolutionary approaches have shown successful results in the former chapters. However, this approach often suffers from the high computational costs which are used for a huge number of performance evaluations. This chapter employs reinforcement learning as an alternative to evolutionary robotics. In general, the designing collective behavior of robotic swarms becomes a challenging problem for reinforcement learning due to the local observability of agents or the dynamical environment by numerous agents. On the other hand, recent trends in the machine learning field have proposed powerful deep reinforcement learning algorithms [50, 54, 101, 129]. Most of these algorithms were benchmarked with single-agent problems; applicability for multi-robot scenarios is unclear. This chapter employs proximal policy optimization [129] which is one of the most popular deep reinforcement learning algorithms to design a controller of a multi-legged robotic swarm.

The rest of this chapter is organized as follows. Section 7.1 briefly describes the PPO algorithm. Section 7.2 shows experimental settings. Section 7.3 describes methods of applying PPO for a multi-legged robotic swarm. Section 7.4 shows the results of the experiments for comparing reward settings. Section 7.5 shows the experiments in rough terrain fields. Finally, Section 7.6 concludes this chapter.

7.1 Proximal Policy Optimization

Proximal Policy Optimization (PPO) is one of the most popular deep reinforcement learning algorithms. The PPO is categorized as a policy gradient method in reinforcement learning. In addition, the PPO usually adopts an actor-critic style. In policy gradient methods, the policy is updated using the gradient of the objective function with respect to the policy parameters. Typically, policy update is represented by the following equation:

$$\theta' = \theta + \eta \nabla_\theta J(\theta) \tag{7.1}$$

where θ is the parameter vector of the policy, $J(\theta)$ is the objective function parameterized by θ, and η is the learning rate of the policy update. In the PPO, the main objective function is the following clipped surrogate objective function:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t \left[\min \left(r_t(\theta)\hat{A}_t, \text{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_t \right) \right] \tag{7.2}$$

where $\hat{A}$ and $\hat{\mathbb{E}}$ denote the empirically obtained estimates of the advantage function and expectation, respectively. The $r_t(\theta)$ is the probability ratio calculated by the following equation:

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} \tag{7.3}$$

where π_θ is the agent policy. s_t and a_t are observed state and selected action of the agent at timestep t, respectively. The θ_{old} is the vector of policy parameters before the update. In Eq. (7.2), ϵ is the hyperparameter that restricts the changes of r_t. The clip term in Eq. (7.2) indicates that the ratio of $\pi_\theta(a_t|s_t)$ to $\pi_{\theta_{\text{old}}}(a_t|s_t)$ is restricted to the range $[1-\epsilon, 1+\epsilon]$. By using the minimum function, the final objective becomes the lower bound of the unclipped objective. The lower bound is also called as *pessimistic bound* [129]. This implies that the relatively large changes in the probability ratio (determined by ϵ) are ignored when the objective is improved. The policy gradient method often leads the policy to destructively large updates. PPO prevents the large policy update by using the clipped surrogate objective function. The basic idea of PPO originated from the trust region policy optimization (TRPO) algorithm [128]. The PPO is a simplified implementation of the TRPO. PPO has succeeded in a variety of control problems, including multi-legged robots [53, 81, 97, 118]. Most studies on PPO have been conducted with a single agent, which means in a static environment. This study employs the PPO to design a robot controller for a multi-legged robotic swarm.

7.2 Settings of Experiments

The aim of this experiment is to generate collective behavior of a multi-legged robotic swarm by using the PPO. The robot controller is trained through the task that requires robots to walk and form a line. The experiment is conducted by computer simulations as in the former chapters.

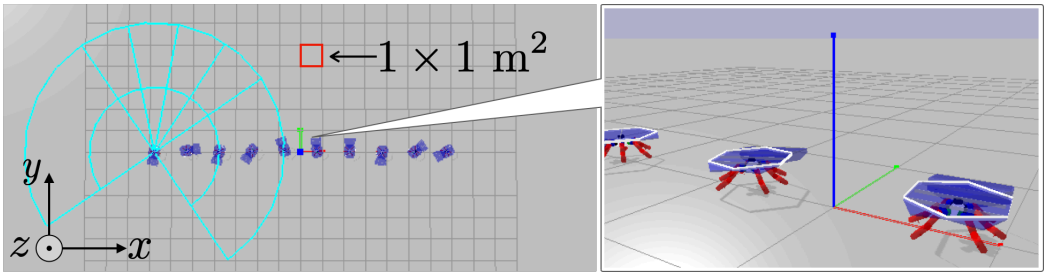


Fig. 7.1. Overview of the task environment. Ten robots are aligned along the x-axis with random directions. The cyan line shows the visible range of the camera equipped with a robot.

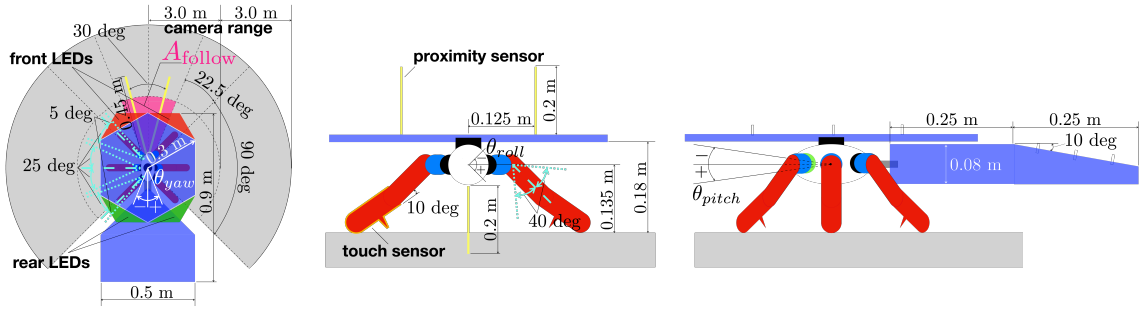


Fig. 7.2. Settings of the robot. Cyan dotted lines indicate the movable range of the joint. The gray circular sector shows the visible range of the camera. The two sections of the visible range ("A_{follow}" in the left figure) are used to calculate the reward for following other robots. Yellow lines are the sensor ranges of the proximity sensors.

7.2.1 Task Settings

The task environment is shown in Fig. 7.1. A total of 10 robots are aligned along the x-axis. The initial directions of robots are determined randomly at the beginning of each episode. One episode consists of 5000 steps (166 s in the physics simulation). In this environment, robots are trained to form a line while walking. Robots are trained to walk in the positive direction of the x-axis. Additionally, robots are also trained to follow other robots and form a line. In this task, robots are required to keep relative positions to other robots using local observation and appropriate gait.

7.2.2 Robot Settings

The robot settings are illustrated in Fig. 7.2. The body structure of the robot is similar to Chapter 4, while the visible range of the camera doubled from Chapter 4. The robot has six legs, each leg consisting of two joints. In Fig. 7.2, the cyan dotted lines indicate the movable ranges of joints. Each robot is also equipped with LEDs, a camera, proximity sensors, touch

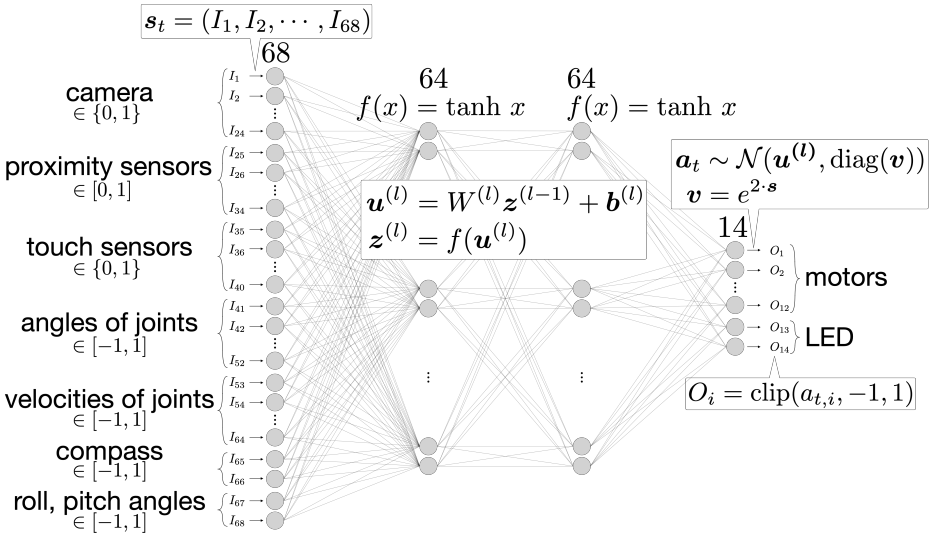


Fig. 7.3. Structure of the robot controller.

sensors, and an electric compass. Two LEDs are attached to the front and rear parts of the robot. The robot controller decides to turn on/off LEDs. The camera is attached to the center of the torso. The visible range of the camera is divided into two sections in the radial direction, and divided into six sections around the angle of the robot, as illustrated in Fig. 7.2. The camera has a total of twelve sections within the visible range. Each section can independently detect the colored LEDs on the front and back of the robot. In total, twenty-four binary signals are obtained from the camera. Additionally, the two sections in front of the robot (A_{follow} in the left figure of Fig. 7.2) are used to calculate the reward for following other robots. Proximity sensors detect other robots or objects within sensor ranges. The values from the proximity sensors are normalized into [0,1]. In this experiment, the proximity sensors are supposed to distinguish between robots and other objects, and return values independently. Touch sensors are equipped at the end of each leg. Each touch sensor returns 1 when it detects collisions with other objects and 0 otherwise. The electric compass outputs sine and cosine values of the direction the robot is facing.

7.3 Methods

The controller of a multi-legged robotic swarm is trained by the PPO algorithm. This section describes the details of a robot controller, a learning algorithm, reward functions, and measurement factors for collective behavior.

7.3.1 Controller

The robot controller (i.e., the policy network) is illustrated in Fig. 7.3. The feedforward neural network with two hidden layers (similar to the structure in [129]) is employed as

the robot controller. The input layer consists of 68 neurons. Inputs to the controller are obtained from a camera, proximity sensors, touch sensors, angles and angular velocities of joints, roll and pitch angles of the torso, and the compass. The output layer consists of 14 neurons. As in [128, 129], the output layer determines the means of Gaussian distributions. The agent action (a_t) is sampled from the distribution with variable standard deviations. The final outputs are obtained by clipping the a_t within the range of $[-1, 1]$. Twelve of outputs determine the target angular velocities of joints. The remaining two decide to turn on/off LEDs. The learnable parameters of the policy network include synaptic weights $W^{(l)}$ in each layer, bias $\boldsymbol{b}^{(l)}$ in each layer, and variance parameter $\boldsymbol{s}$ in the output layer. The value network has similar a structure to the policy network except for the output layer. The output layer of the value network has a single neuron that predicts the value $V(s_t)$.

Algorithm 3 PPO algorithm for a robotic swarm

```
for iteration=1, 2, ... do
    for actor=1, 2, ..., N do
        Run policy π_θold in environment for T timesteps
        Compute advantage estimates Â_1, ..., Â_T based on a robot i
        if an episode is the end then
            Re-select the robot i from a robotic swarm randomly
        end if
    end for
    Optimize surrogate L wrt θ, with K epochs and minibatch size M ≤ NT
    θ_old ← θ
end for
```

7.3.2 Learning Algorithm

The robot controller is trained by the PPO algorithm. In this study, the learning objective is set as follows:

$$L_t^{CLIP+VF} = \hat{\mathbb{E}}_t \left[L_t^{CLIP}(\theta) - L_t^{VF}(\theta) \right]. \tag{7.4}$$

The $L^{CLIP}(\theta)$ in Eq. (7.2) is employed as $L_t^{CLIP}(\theta)$ without any changes. The $L_t^{VF}(\theta)$ is the squared-error loss of the value function $(V_\theta(s_t) - V_t^{\text{targ}})^2$. The notation of Eq. (7.4) is based on [129].

The PPO algorithm often employs parallel environments (actors) for sampling data. In each environment, data for learning (such as an observation, selected action, reward, and next state) are sampled from a single agent. In a robotic swarm scenario, each environment includes numerous robots (agents). In this study, a single robot in a robotic swarm is randomly selected for sampling data. Therefore, the original PPO can be applied to train

a robotic swarm without any changes. On the other hand, the action of each robot is determined based on local observation (i.e., without using the other robot's observation). This means all robots have the same policy independently. This method is an example of a "centralized training with decentralized execution" framework in multi-agent reinforcement learning [90]. The pseudocode is presented in Algorithm 3. This study employs the PFRL library [41] for PPO implementation. The parameter setup of the algorithm is summarized in Appendix B. The values of parameters are tuned by preliminary experiments. The most important factor for successful learning in the dynamic environment seems a small clipping ϵ. The small ϵ realizes stable learning, while the learning process progressed more slowly.

7.3.3 Reward Settings

This study employs two types of rewards: r_1 for walking and r_2 for following other robots. The r_1 is represented by the following equation:

$$r_1 = K_1 \cdot f_{v_x} \cdot \prod_{k=1}^{5} \delta_k \tag{7.5}$$

where K_1 is the constant value for scaling, and f_{v_x} is the function of the robot velocity. The f_{v_x} is represented as follows:

$$f_{v_x} = \min\left(v_{\max},\ av_x,\ a(2v_{\max} - v_x)\right) \tag{7.6}$$

where a and $v_{\max}$ are constants. The $v_{\max}$ (m/step) is the target speed of walking robots. The v_x is the robot velocity along the x-axis. The reward based on v_x is inspired by benchmarks in MuJoCo.[*2] The f_v has a trapezoid shape that is centered at $v_{\max}$. In Eq. 7.5, δ_1 to δ_5 are supplemental parts to make a robot's gait similar to natural organisms, as in Chapter 4. The details of δ_1 to δ_5 are described in Appendix A. Finally, the r_1 requires robots to walk in a positive direction of the x-axis at the approximate speed $v_{\max}$.

The r_2 is represented as follows:

$$r_2 = K_2 \cdot r_{\text{follow}} \tag{7.7}$$

$$r_{\text{follow}} = \begin{cases} 1 & \text{if colored LEDs are detected in } A_{\text{follow}} \text{ of the camera} \\ 0 & \text{otherwise} \end{cases}$$

where K_2 is the constant value for scaling. The value of r_{follow} is determined by the sensing results of A_{follow} in Fig. 7.2. The mechanism of r_2 is also similar to $fitness_2$ in Chapter 4. The r_2 requires robots to follow other robots and maintain a line formation.

The total reward is designed by combining r_1 and r_2. In the experiment, four reward settings are prepared as follows:

[*2] Available at https://www.gymlibrary.dev/environments/mujoco/.

$$\textit{Setting A: } r = r_1$$
$$\textit{Setting B: } r = r_1 + r_2$$
$$\textit{Setting C: } r = \begin{cases} r_1 & \text{if episode} < E_{r\_\text{change}} \\ r_1 + r_2 & \text{otherwise} \end{cases}$$
$$\textit{Setting D: } r = r_1 + g(r_1) \cdot r_2$$

where r denotes the total reward for robots. Setting A is prepared to check the effect of r_2. Setting B is the simple implementation for using both r_1 and r_2. Setting C and Setting D employ hierarchical learning processes about r_1 and r_2. In Setting C, the r is changed based on the elapsed learning steps. The E_{r_change} is the episode in which the reward function is switched. In Setting D, r_2 is multiplied by the function $g(r_1)$. The $g(x)$ is given as follows:

$$g(x) = \begin{cases} 1 & x \geq \alpha \\ 0 & \text{otherwise} \end{cases} \tag{7.8}$$

where α is the constant value. The $g(x)$ is a kind of step function that uses α as a threshold. By using the $g(r_1)$, r_2 becomes available after r_1 is trained to a certain extent. This approach is based on a similar idea to the constrained optimization methods [147]. The parameter settings of the reward functions are summarized in Appendix B.

7.3.4 Measurement Factors for Collective Behavior

Measurement factors are employed to evaluate the achievement of the task (i.e., whether robots form a line while they are walking). The remaining part of this section describes details of measurement factors.

1. The $N_{\text{Largest\_Cluster}}$ is the number of robots that belong to the largest cluster [3, 150]. Robot i and j ($i \neq j$) are regarded as "connected" if the distance between robot i and j is shorter than the threshold (in this study, the threshold is set as 4.5 m). The connections among robots are represented by an adjacency matrix. If there is a path from robot i to j, they belong to the same cluster (paths among robots are represented by a reachability matrix). The $N_{\text{Largest\_Cluster}}$ detects whether robots keep distances relatively close to other robots.
2. The $\bar{x}$ is the mean x-coordinate value of robots that belong to the largest cluster. The $\bar{x}$ detects whether the group of robots moves along the x-axis.
3. The d_{Major} and d_{Minor} are the major and minor diameters of the minimal ellipse (Löwner-John ellipsoid [65]) that includes the largest cluster. The ellipse shows the approximated shape of the largest cluster to detect whether robots form a line.

Based on these metrics, the desirable behavior should show: (1) a high value of $N_{\text{Largest\_Cluster}}$, (2) increasing $\bar{x}$, (3) a high value of d_{Major}, and (4) a low value of d_{Minor}.

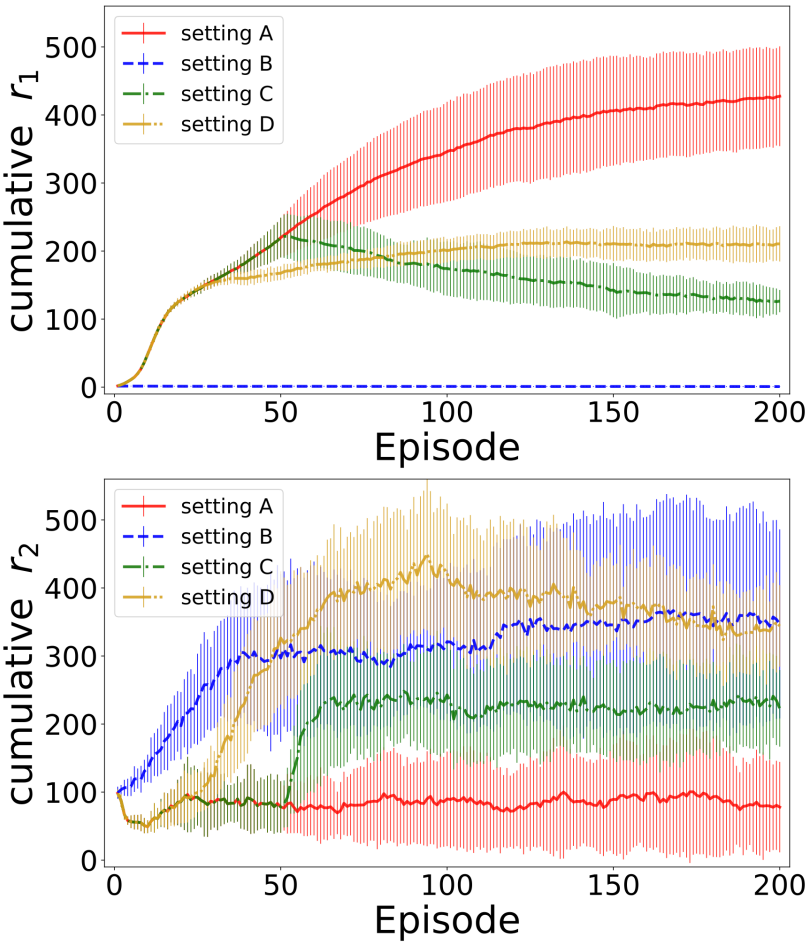


Fig. 7.4. Reward transition through the learning process. In each figure, plots are averaged over 10 robots, 16 parallel environments, and 10 learning trials. The standard deviation is calculated for 10 learning trials. Note that r_2 is not used as the reward signal in Setting A.

7.4 Experiments for Comparing Reward Settings

This section describes experiments for comparing reward settings in Section 7.3.3. In each reward setting, a total of 10 learning trials are conducted. The reward transitions during the learning process are shown in Fig. 7.4. The above plots of Fig. 7.4 show that Setting A exhibits the best performance for obtaining r_1. In contrast, r_1 does not increase in Setting B. This indicates that robots in Setting B do not walk well in the environment. This phenomenon seems to originate from the reward scales or the fact that r_2 is denser than r_1. The plots also show that the cumulative r_1 in Setting C decreased after the episode E_{r_change}, whereas r_1 in Setting D increased consistently. This indicates that Setting D shows a better performance of r_1 than Setting C. Additionally, the below plots of Fig. 7.4 show that Setting D recorded the best performance of r_2 around the 90th episode. Setting D also showed better performance for obtaining r_2 than Setting C.

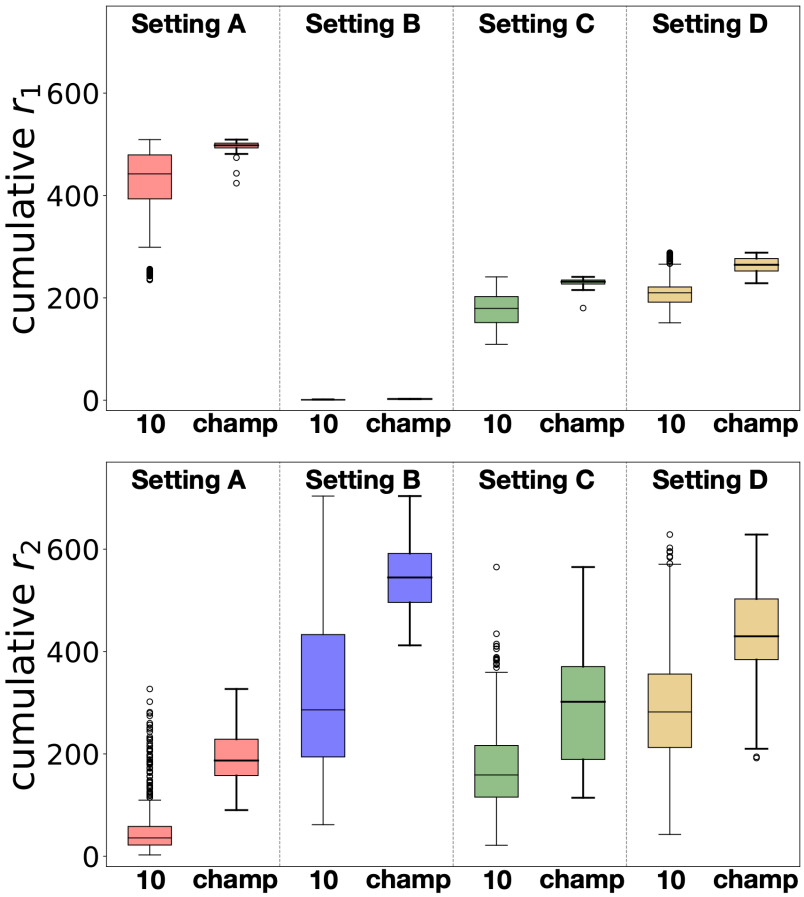


Fig. 7.5. Results of re-evaluation for obtained controllers. Boxes of the notation "10" show the results of the best controllers in each of the 10 learning trials (each box consists of 500 plots). The "champ" means the controller that shows the best performance among the left box (each box consists of 50 plots).

For statistical comparison, obtained controllers are re-evaluated. The controller with the best performance was recorded in each learning trial. A total of 10 controllers were obtained from each setting. Each controller was re-evaluated 50 times in the task. The results of re-evaluations are summarized in Fig. 7.5. The results from Fig. 7.5 confirm that the tendency of obtained rewards among settings is similar to Fig. 7.4 (e.g., Setting A is the best for r_1, and Setting D shows better performance than Setting C).

Examples of observed behaviors are shown in Fig. 7.6. The robot behaviors are generated by using the "champ" controllers in Fig. 7.5. Fig. 7.6 shows that robots in Setting A spread out while walking because Setting A does not use r_2. In Setting B, robots almost stayed at the initial positions because controllers primarily received r_2. In Setting C and D, robots succeeded in forming a line while moving in the right direction. Additionally, Fig. 7.7 shows more long-term results for obtained behaviors as robot trajectories. Fig. 7.7 shows that the

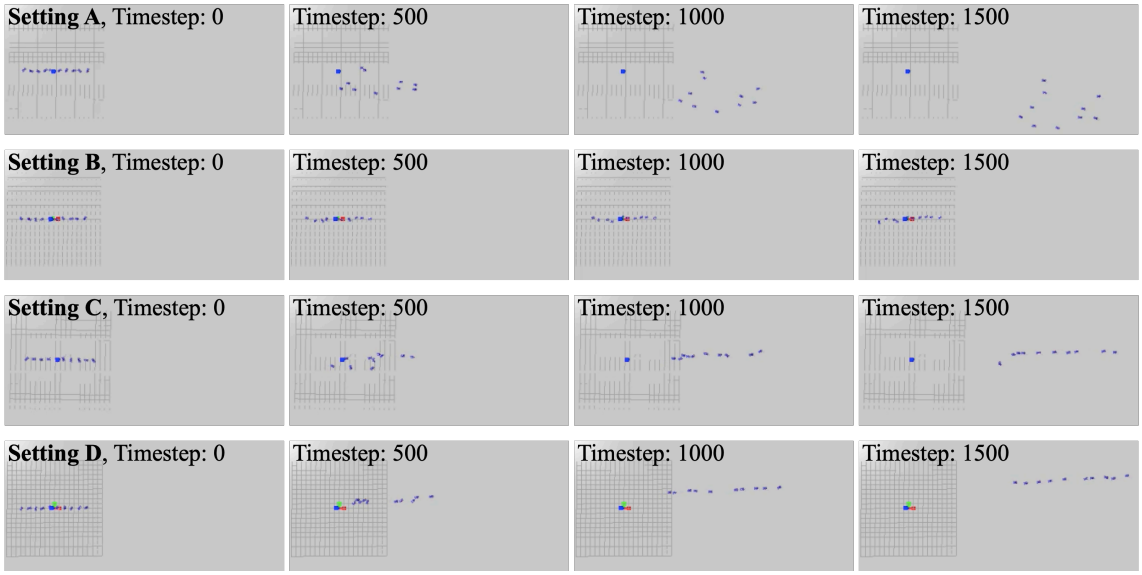


Fig. 7.6. Examples of observed behaviors. The behavior is generated by the "champ" controller from each reward setting.

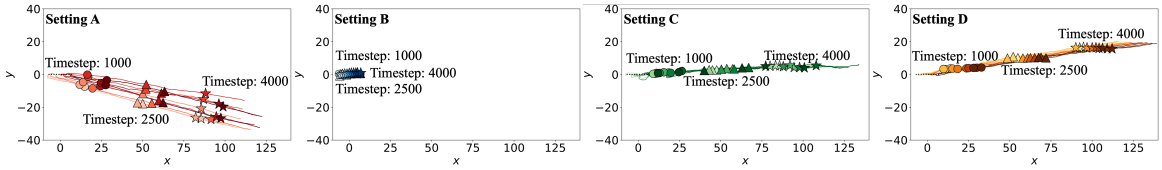


Fig. 7.7. Robot trajectories over 5000 timesteps in the task. The circle, triangle, and star makers show the robot positions at 1000, 2500, and 4000 timesteps, respectively.

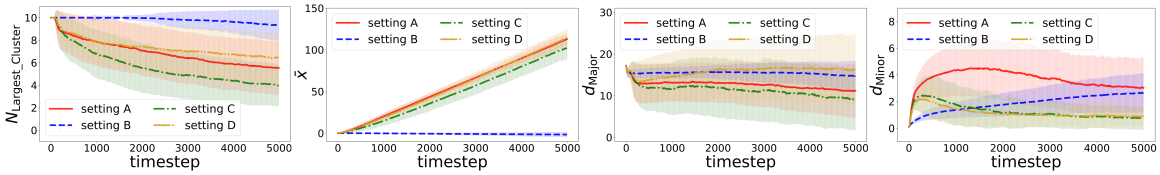


Fig. 7.8. Transitions of measurement factors through the task period. These are the results of the best controllers in each of the 10 learning trials in Fig. 7.5.

behaviors shown in Fig. 7.6 continued until the end of the task. These results showed that Setting C and Setting D succeeded in maintaining the line formations of robots.

The transitions of measurement factors are summarized in Fig. 7.8. The measurement factors show the statistical aspects of robot behaviors. Fig. 7.8 shows that Setting D seems to have the best performance for achieving the task (a high value of $N_{\text{Largest\_Cluster}}$, increasing $\bar{x}$, a high value of d_{Major}, and a low value of d_{Minor}). The results from Fig. 7.4 to Fig. 7.8 show that the PPO algorithm successfully designed how to coordinate joints on legs to form

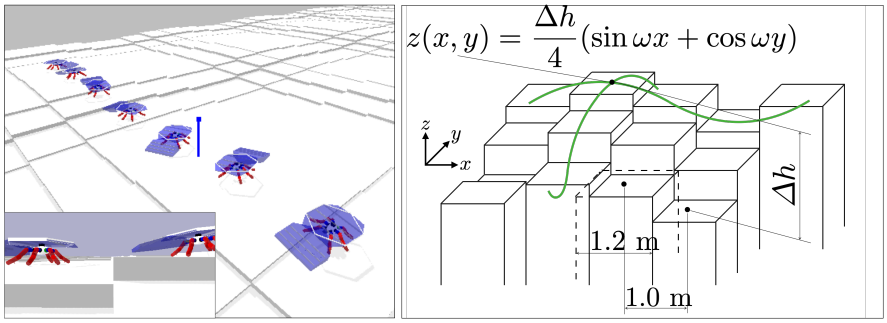


Fig. 7.9. Setting of the rough terrain field. The field consists of cuboid blocks that form a sinusoidal surface.

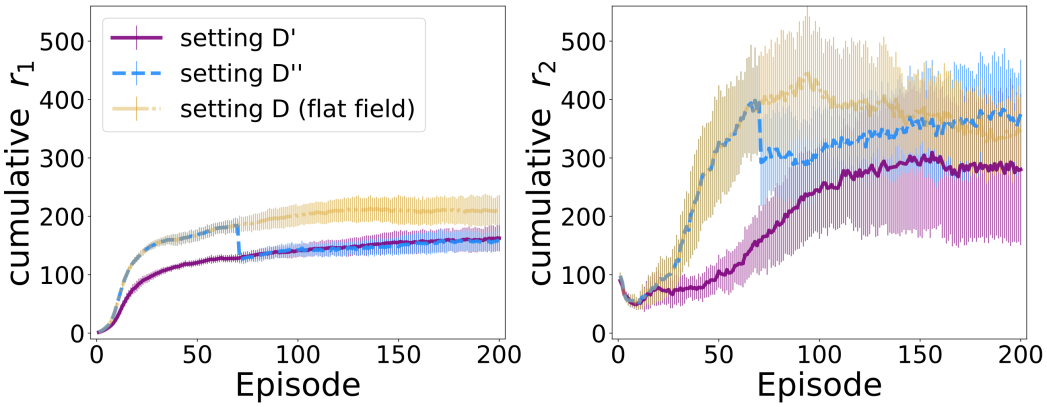
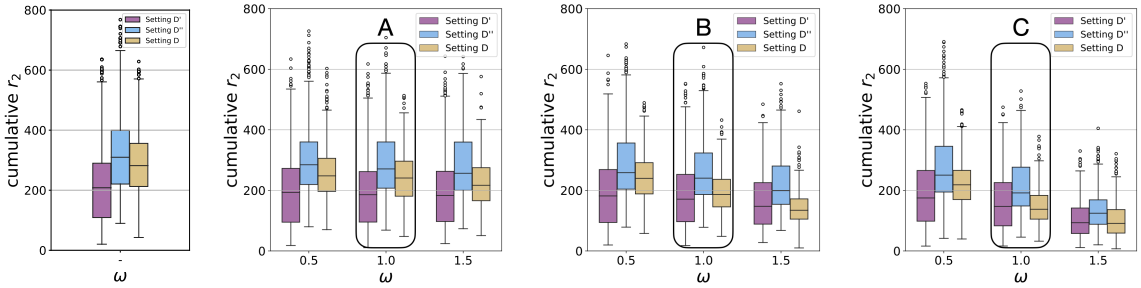


Fig. 7.10. Reward transitions in the rough terrain field.



(a) $\Delta h = 0.0$. **(b)** $\Delta h = 0.15$. **(c)** $\Delta h = 0.3$. **(d)** $\Delta h = 0.45$.

Fig. 7.11. Boxplots of r_2 for varied Δh and ω in the rough terrain field. These are results of the best controllers in each of the 10 learning trials.

a line based on local observations of robots. The results also indicate that the hierarchical learning for rewards is effective for training the multi-legged robotic swarm.

7.5 Experiments on Rough Terrain

This section presents the experiments conducted in the rough terrain field. The multi-legged robotic swarm is expected to operate not only on the flat surface but also on rough terrains. Fig. 7.9 shows the environmental settings. The rough terrain field consists of cuboid blocks that form a sinusoidal surface. The terrain settings are similar to Chapter 5. The Δh is the height difference between the highest and the lowest points. The ω is the frequency of a sine wave. In the learning process, Δh and ω are set as $(\Delta h, \omega) = (0.3, 1.0)$. Blocks are placed in $\{(x, y)| -30 \leq x \leq 30, -10 \leq y \leq 150\}$.

During the experiments in Section 7.4, Setting D exhibited the best performance for achieving the task. Therefore, learning processes based on Setting D are conducted in the rough terrain field. The settings of experiments are designed as follows:

Setting D': Start training on the rough terrain field with the reward of Setting D

Setting D": Start training on the flat field and switch to the rough terrain field (the reward setting is also Setting D).

These settings verify the effects of environmental transitions on the learning process. Setting D" employs hierarchical training from flat to rough terrain. In Setting D", the rough terrain field is used after the 70th episode based on the results of Section 7.4. In both settings, the same parameter settings as Section 7.4 are used for learning.

The reward transitions during the learning process are summarized in Fig. 7.10. The results of Setting D in Section 7.4 are also plotted for comparison. In Fig. 7.10, the r_1 of Setting D" decreased when changing the flat field to the rough terrain field. The r_2 of Setting D" also decreased when changing the field. However, Setting D" shows a better performance of r_2 than Setting D'.

The controllers obtained from the rough terrain field were applied to various terrain settings to check the controller's adaptability. Additionally, controllers of Setting D (trained in the flat field) were also tested on rough terrains. The Δh was varied as $\{0.0, 0.15, 0.3, 0.45\}$. The ω was also varied as $\{0.5, 1.0, 1.5\}$. A total of 10 terrain settings (ω is irrelevant when Δh = 0.0) were employed. Fig. 7.11 summarizes the results of cumulative r_2 for varied terrain settings. Fig. 7.11 indicates that the cumulative r_2 decreased as the Δh and ω increased. This tendency is similar to the results in Chapter 5.

Fig. 7.12 shows the examples of robot trajectories in the field of $(\Delta h, \omega) = (0.15, 1.0)$ ("A" in Fig 7.11(b)). In Fig. 7.12, the right plots show that robots of Setting D (trained on the flat field) can maintain a line on the rough terrain field. Additionally, transitions of measurement factors are summarized in Fig. 7.13. The $\bar{x}$ of Setting D showed worse performance than

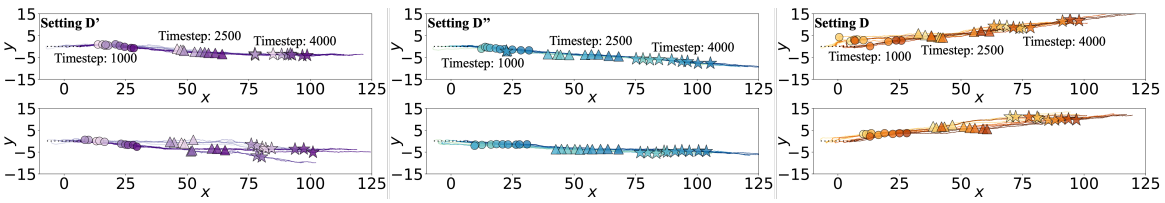


Fig. 7.12. Examples of robot trajectories at $(\Delta h, \omega) = (0.15, 1.0)$ ("A" in Fig 7.11(b)). The circle, triangle, and star makers show the robot positions at 1000, 2500, and 4000 timesteps, respectively.

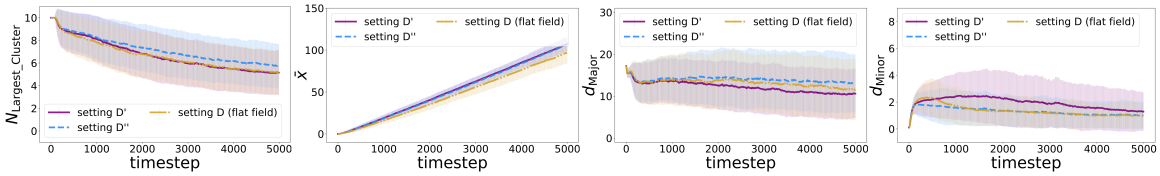


Fig. 7.13. Transitions of measurement factors at $(\Delta h, \omega) = (0.15, 1.0)$ ("A" in Fig 7.11(b)).

other settings. However, the remaining measurement factors of Setting D showed similar or better performances than others. These results indicate that the controllers trained on the flat field can cope with the rough terrain field that is relatively close to the flat (i.e., small Δh and ω).

Fig. 7.14 shows the examples of robot trajectories in the field of $(\Delta h, \omega) = (0.30, 1.0)$ ("B" in Fig 7.11(c)). The transitions of measurement factors are also summarized in Fig. 7.15. Compared with Fig. 7.12, the right plots of Fig. 7.14 show that it is difficult for robots of Setting D to maintain a line. Fig. 7.15 also shows that the performance in Setting D is inferior to other settings. On the other hand, the center of Fig. 7.14 shows that the robots of Setting D" maintain lines in the terrain setting where the learning process was conducted. The result of Fig 7.10 and Fig. 7.14 indicates that the PPO algorithm also succeeded in training a robot controller in the rough terrain field.

Fig. 7.16 and Fig. 7.17 show the results in $(\Delta h, \omega) = (0.45, 1.0)$ ("C" in Fig 7.11(d)). This terrain setting makes it difficult for all of the controllers to maintain a line. However, Fig. 7.17 shows that Setting D' and Setting D" perform better than Setting D. This indicates that the controllers trained in the rough terrain field exhibited higher adaptability for rougher terrain settings.

7.6 Conclusions

This chapter demonstrated a collective behavior of a multi-legged robotic swarm by using deep reinforcement learning. The proximal policy optimization (PPO) algorithm was employed as an alternative to the evolutionary robotics approach. The PPO algorithm was

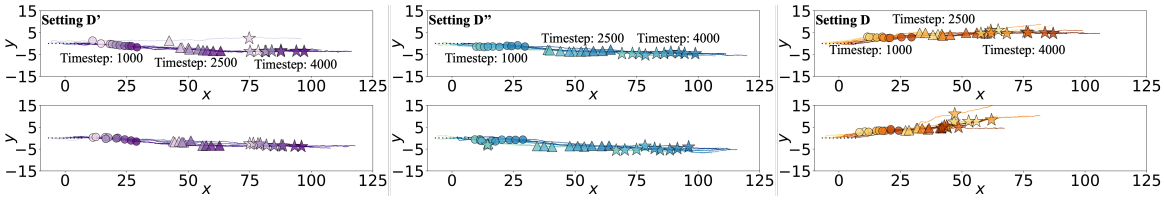


Fig. 7.14. Examples of robot trajectories at $(\Delta h, \omega) = (0.30, 1.0)$ ("B" in Fig 7.11(c)).

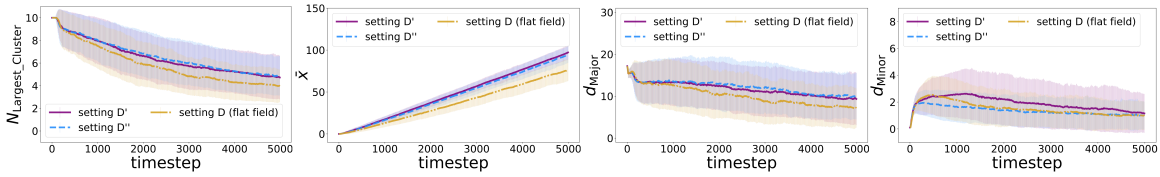


Fig. 7.15. Transitions of measurement factors at $(\Delta h, \omega) = (0.30, 1.0)$ ("B" in Fig 7.11(c)).

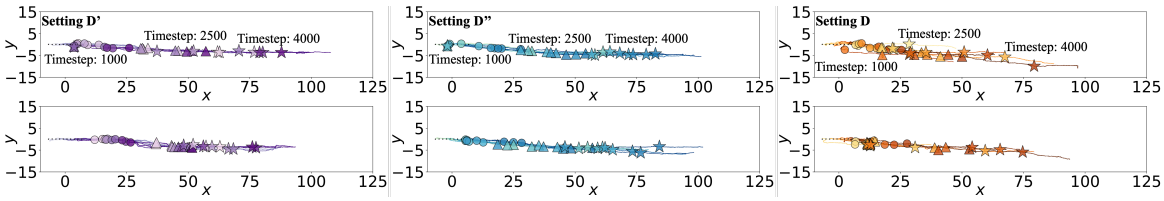


Fig. 7.16. Examples of robot trajectories at $(\Delta h, \omega) = (0.45, 1.0)$ ("C" in Fig 7.11(d)).

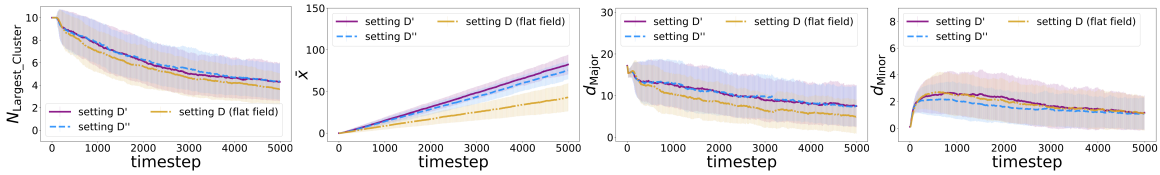


Fig. 7.17. Transitions of measurement factors at $(\Delta h, \omega) = (0.45, 1.0)$ ("C" in Fig 7.11(d)).

slightly extended for the multi-robot scenario. The results of computer simulations showed that the proposed algorithm and reward functions have succeeded in generating collective behavior of a multi-legged robotic swarm as evolutionary methods.

Chapter 8

Conclusions

This thesis presented automatic designs of controllers for a multi-legged robotic swarm. Multi-legged robotic swarms are expected to operate in rough terrains or show novel collective behavior inspired by army ants. However, designing controllers becomes a challenging problem because a controller decides not only how to coordinate a large number of robots, but also how to coordinate a large number of actuators in individual robots. Therefore, multi-legged robotic swarms raise a combined problem between two types of large-degree-of-freedom controls. To solve this problem, this thesis focused on the potential of automatic design methods. In swarm robotics, automatic design methods are divided into two main domains: evolutionary robotics and reinforcement learning. This thesis employed both approaches to generate collective behaviors of a multi-legged robotic swarm. This thesis contributes to the swarm robotics community by following two aspects.

First, this thesis presented evolutionary robotics approaches for designing controllers of a multi-legged robotic swarm. Chapter 3 proposed the method of combining the perceptron with the central pattern generator (CPG). The evolutionary robotics approach was applied to the synaptic weights of the perceptron. The experimental results showed that the proposed method successfully designed a collective step-climbing behavior. Chapter 4 provided the pure neuroevolution approach instead of the hand-designed CPG. In this chapter, the evolution process with the proposed fitness function succeeded in generating robot gaits similar to natural organisms. Additionally, the neuroevolution approach generated a basic collective behavior of a robotic swarm. In Chapter 5, multi-legged robots are applied to the path formation task on rough terrains. The performance of a robotic swarm was discussed with the terrain settings. The results showed that the incremental evolution was effective to design a collective behavior in rough terrains. Chapter 6 focused on generating and analyzing collective step-climbing behavior. The measurement factors were proposed to examine what kind of behaviors were obtained through the evolution process. The results indicated the robots act as stepping stones that contribute to achieving the task. These results showed

that the evolutionary robotics approach was promising to design collective behaviors of a multi-legged robotic swarm.

Second, this thesis presented the reinforcement learning-based approach as an alternative to evolutionary robotics. Reinforcement learning is a common approach to obtaining an agent's action, while it often suffers from dynamic environmental settings with multiple agents. In Chapter 7, the proximal policy optimization (PPO) algorithm was slightly extended for a multi-agent scenario. Experimental results showed that the PPO succeeded in generating collective behavior of a multi-legged robotic swarm. The results also indicated that the performance of a robotic swarm on rough terrains had a similar tendency to the evolutionary robotics approach. Chapter 7 implied the potential of reinforcement learning for more advanced tasks.

8.1 Future Work

First, one of the future directions for swarm robotics is developing real-world applications [28]. So far, automatic design methods have shown promising results for designing robot controllers. Also in this thesis, the automatic design methods showed great potential for designing sophisticated controllers (i.e., coordinating a large number of joints for generating collective behavior) in physics simulations. However, physics engines do not emulate all of the properties of real-world phenomena; therefore, the *reality gap* [63, 134] is still a critical problem. To overcome this issue and operate robotic swarms in the real world, it important to focuses on promising topics such as digital twins [43] or cyber-physical systems [102].

Another advanced direction is discussing interactions between two types of algorithms: evolutionary robotics and reinforcement learning. This thesis showed the potential of both methods for a multi-legged robotic swarm, while not yet achieving fair comparison. In robotic swarm problems, these methods typically showed a trade-off between computational cost and performance; evolutionary robotics is computationally expensive but shows better performances, and vice versa. Hybridization of both methods becomes the next direction for establishing more powerful algorithms [4, 74]. In addition, it is insightful to discuss both algorithms with biological terminologies such as *Lamarckism* or *Baldwin effect* [32].

Finally, for more fundamental and general goals, it is time to discuss the methodology of *how to coordinate a large number of elements to realize desirable properties in the system.* Obviously, most things in the real world can be regarded as the composite of smaller elements (f.g., cells making creatures consist of smaller elements such as molecules or atoms). In the engineering field, the artifacts consisting of numerous components involve a large degree of freedom; these systems are expected to operate in a wide range of environmental conditions (f.g., redundant robots [18, 133] or soft robots [84, 131]). Generally, it is highly difficult to predict or control the aspect of the system which consists of enormous elements [57, 89, 139].

On the other hand, the coordination of a huge number of elements has shown remarkable results in the recent machine learning field [46]. Based on these ideas, one of the future directions related to this thesis is like a morphogenetic evolution [21, 76]; discuss how to build a unit robot of the swarm from primitive components. Additionally, this approach also needs to discuss the transferability to the real world [79, 116].

References

[1] Anderson, C., Theraulaz, G., and Deneubourg, J.-L. Self-assemblages in insect societies. *Insectes sociaux*, Vol. 49, No. 2, pp. 99–110, 2002.

[2] Badia, A. P., Piot, B., Kapturowski, S., Sprechmann, P., Vitvitskyi, A., Guo, Z. D., and Blundell, C. Agent57: Outperforming the atari human benchmark. In *International conference on machine learning*, pp. 507–517. PMLR, 2020.

[3] Bahgeçi, E. and Sahin, E. Evolving aggregation behaviors for swarm robotic systems: A systematic case study. In *Proceedings 2005 IEEE Swarm Intelligence Symposium, 2005. SIS 2005.*, pp. 333–340. IEEE, 2005.

[4] Bai, H., Cheng, R., and Jin, Y. Evolutionary reinforcement learning: A survey. *Intelligent Computing*, Vol. 2, p. 0025, 2023.

[5] 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.

[6] Bellman, R. A markovian decision process. *Journal of Mathematics and Mechanics*, pp. 679–684, 1957.

[7] Beni, G. From swarm intelligence to swarm robotics. In *Swarm Robotics: SAB 2004 International Workshop, Santa Monica, CA, USA, July 17, 2004, Revised Selected Papers 1*, pp. 1–9. Springer, 2005.

[8] Beni, G. and Wang, J. Swarm intelligence in cellular robotic systems. In *Robots and biological systems: towards a new bionics?*, pp. 703–712. Springer, 1993.

[9] Berlinger, F., Gauci, M., and Nagpal, R. Implicit coordination for 3d underwater collective behaviors in a fish-inspired robot swarm. *Science Robotics*, Vol. 6, No. 50, p. eabd8668, 2021.

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

[11] Bhushan, B. Biomimetics: lessons from nature–an overview. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, Vol. 367, No. 1893, pp. 1445–1486, 2009.

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

[13] Boston Dynamics. Spot. `https://www.bostondynamics.com/products/spot`.

[14] 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.

[15] Brooks, R. A. A robot that walks; emergent behaviors from a carefully evolved network. *Neural computation*, Vol. 1, No. 2, pp. 253–262, 1989.

[16] Chakraborty, A. and Kar, A. K. Swarm intelligence: A review of algorithms. *Nature-inspired computing and optimization: Theory and applications*, pp. 475–494, 2017.

[17] Chen, Y., Zhao, H., Mao, J., Chirarattananon, P., Helbling, E. F., Hyun, N.-s. P., Clarke, D. R., and Wood, R. J. Controlled flight of a microrobot powered by soft artificial muscles. *Nature*, Vol. 575, No. 7782, pp. 324–329, 2019.

[18] Chirikjian, G. S. and Burdick, J. W. The kinematics of hyper-redundant robot locomotion. *IEEE transactions on robotics and automation*, Vol. 11, No. 6, pp. 781–793, 1995.

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

[20] Clune, J., Beckmann, B. E., Ofria, C., and Pennock, R. T. Evolving coordinated quadruped gaits with the hyperneat generative encoding. In *2009 iEEE congress on evolutionary computation*, pp. 2764–2771. IEEE, 2009.

[21] Corucci, F., Cheney, N., Kriegman, S., Bongard, J., and Laschi, C. Evolutionary developmental soft robotics as a framework to study intelligence and adaptive behavior in animals and plants. *Frontiers in Robotics and AI*, Vol. 4, p. 34, 2017.

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

[23] 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.

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

[25] Dorigo, M., Birattari, M., Blum, C., Clerc, M., Stützle, T., and Winfield, A. *Ant Colony Optimization and Swarm Intelligence: 6th International Conference, ANTS 2008, Brussels, Belgium, September 22-24, 2008, Proceedings*, Vol. 5217. Springer, 2008.

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

[27] Dorigo, M., Theraulaz, G., and Trianni, V. Reflections on the future of swarm robotics. *Science Robotics*, Vol. 5, No. 49, p. eabe4385, 2020.

[28] 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.

[29] Dorigo, M., Trianni, V., Şahin, E., Groß, R., Labella, T. H., Baldassarre, G., Nolfi, S.,

Deneubourg, J.-L., Mondada, F., Floreano, D., et al. Evolving self-organizing behaviors for a swarm-bot. *Autonomous Robots*, Vol. 17, No. 2-3, pp. 223–245, 2004.

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

[31] Eriksson, A., Nilsson Jacobi, M., Nyström, J., and Tunstrøm, K. Determining interaction rules in animal swarms. *Behavioral Ecology*, Vol. 21, No. 5, pp. 1106–1111, 2010.

[32] Fernando, C., Sygnowski, J., Osindero, S., Wang, J., Schaul, T., Teplyashin, D., Sprechmann, P., Pritzel, A., and Rusu, A. Meta-learning by the baldwin effect. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pp. 1313–1320. 2018.

[33] Festo. Bionicswift. `https://www.festo.com.cn/cn/en/e/about-festo/research-and-development/bionic-learning-network/highlights-from-2018-to-2021/bionicswift-id_326830/`.

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

[35] Floreano, D., Husbands, P., and Nolfi, S. Evolutionary robotics. *Tech. rep.*, Springer Verlag, 2008.

[36] 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.

[37] 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.

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

[39] Francesca, G., Brambilla, M., Brutschy, A., Garattoni, L., Miletitch, R., Podevijn, G., Reina, A., Soleymani, T., Salvaro, M., Pinciroli, C., et al. Automode-chocolate: automatic design of control software for robot swarms. *Swarm Intelligence*, Vol. 9, No. 2, pp. 125–152, 2015.

[40] Fujimoto, S., Hoof, H., and Meger, D. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pp. 1587–1596. PMLR, 2018.

[41] Fujita, Y., Nagarajan, P., Kataoka, T., and Ishikawa, T. Chainerrl: A deep reinforcement learning library. *Journal of Machine Learning Research*, Vol. 22, No. 77, pp. 1–14, 2021. URL `http://jmlr.org/papers/v22/20-376.html`.

[42] Fukuhara, A., Gunji, M., Masuda, Y., Tadakuma, K., and Ishiguro, A. Flexible shoulder in quadruped animals and robots guiding science of soft robotics. *Journal of Robotics and Mechatronics*, Vol. 34, No. 2, pp. 304–309, 2022.

[43] 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.

[44] Gauci, M., Chen, J., Dodd, T. J., and Groß, R. Evolving aggregation behaviors in multi-robot systems with binary sensors. In Ani Hsieh, M. and Chirikjian, G. (Eds.), *Distributed Autonomous Robotic Systems*, Vol. 104 of *Springer Tracts in Advanced Robotics*, pp. 355–367. Springer, 2014.

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

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

[47] Groβ, R. and Dorigo, M. Evolution of solitary and group transport behaviors for autonomous robots capable of self-assembling. *Adaptive Behavior*, Vol. 16, No. 5, pp. 285–305, 2008.

[48] 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.

[49] Haarnoja, T., Ha, S., Zhou, A., Tan, J., Tucker, G., and Levine, S. Learning to walk via deep reinforcement learning. *arXiv preprint arXiv:1812.11103*, 2018.

[50] Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P., et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018.

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

[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] Heess, N., TB, D., Sriram, S., Lemmon, J., Merel, J., Wayne, G., Tassa, Y., Erez, T., Wang, Z., Eslami, S., et al. Emergence of locomotion behaviours in rich environments. *arXiv preprint arXiv:1707.02286*, 2017.

[54] Hessel, M., Modayil, J., Van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M., and Silver, D. Rainbow: Combining improvements in deep reinforcement learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*. 2018.

[55] 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.

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

[57] Holland, J. H. *Emergence: From chaos to order*. OUP Oxford, 2000.

[58] Horgan, D., Quan, J., Budden, D., Barth-Maron, G., Hessel, M., Van Hasselt, H., and Silver, D. Distributed prioritized experience replay. *arXiv preprint arXiv:1803.00933*, 2018.

[59] Hornby, G. S., Takamura, S., Yamamoto, T., and Fujita, M. Autonomous evolution of dynamic gaits with two quadruped robots. *IEEE transactions on Robotics*, Vol. 21, No. 3, pp. 402–410, 2005.

[60] 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.

[61] Inagaki, S., Yuasa, H., and Arai, T. Cpg model for autonomous decentralized multi-legged robot system?generation and transition of oscillation patterns and dynamics of oscillators. *Robotics and Autonomous Systems*, Vol. 44, No. 3-4, pp. 171–179, 2003.

[62] Jain, A. K., Mao, J., and Mohiuddin, K. M. Artificial neural networks: A tutorial. *Computer*, Vol. 29, No. 3, pp. 31–44, 1996.

[63] 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.

[64] Jing, G., Tosun, T., Yim, M., and Kress-Gazit, H. An end-to-end system for accomplishing tasks with modular robots. In *Robotics: Science and systems*, Vol. 2, p. 7. 2016.

[65] John, F. Extremum problems with inequalities as subsidiary conditions. In *Traces and emergence of nonlinear programming*, pp. 197–215. Springer, 2014.

[66] Kaelbling, L. P., Littman, M. L., and Moore, A. W. Reinforcement learning: a survey. *Journal of Artificial Intelligence Research*, Vol. 4, pp. 237–285, 1996.

[67] Kakade, S. M. A natural policy gradient. *Advances in neural information processing systems*, Vol. 14, 2001.

[68] Kamegawa, T., Yarnasaki, T., Igarashi, H., and Matsuno, F. Development of the snake-like rescue robot" kohga". In *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004*, Vol. 5, pp. 5081–5086. IEEE, 2004.

[69] Karaboga, D. and Akay, B. A comparative study of artificial bee colony algorithm. *Applied mathematics and computation*, Vol. 214, No. 1, pp. 108–132, 2009.

[70] Katzschmann, R. K., DelPreto, J., MacCurdy, R., and Rus, D. Exploration of underwater life with an acoustically controlled soft robotic fish. *Science Robotics*, Vol. 3, No. 16, p. eaar3449, 2018.

[71] Kennedy, J. *Swarm intelligence*. Springer, 2006.

[72] Kennedy, J. Particle swarm optimization. In *Encyclopedia of machine learning*, pp. 760–766. Springer, 2011.

[73] 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.

[74] Khadka, S. and Tumer, K. Evolution-guided policy gradient in reinforcement learning. *Advances in Neural Information Processing Systems*, Vol. 31, 2018.

[75] Kida, T., Sueoka, Y., Shigeyoshi, H., Tsunoda, Y., Sugimoto, Y., and Osuka, K. Verification of acoustic-wave-oriented simple state estimation and application to swarm navigation. *Journal of Robotics and Mechatronics*, Vol. 33, No. 1, pp. 119–128, 2021.

[76] Koike, R., Ariizumi, R., and Matsuno, F. Automatic robot design inspired by evolution of vertebrates. *Artificial Life and Robotics*, Vol. 27, No. 4, pp. 624–631, 2022.

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

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

[79] Kriegman, S., Blackiston, D., Levin, M., and Bongard, J. A scalable pipeline for designing reconfigurable organisms. *Proceedings of the National Academy of Sciences*, Vol. 117, No. 4, pp. 1853–1859, 2020.

[80] Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pp. 1097–1105. 2012.

[81] Kumar, A., Fu, Z., Pathak, D., and Malik, J. Rma: Rapid motor adaptation for legged robots. *arXiv preprint arXiv:2107.04034*, 2021.

[82] Landgraf, T., Nguyen, H., Forgo, S., Schneider, J., Schröer, J., Krüger, C., Matzke, H., Clément, R. O., Krause, J., and Rojas, R. Interactive robotic fish for the analysis of swarm behavior. In *Advances in Swarm Intelligence: 4th International Conference, ICSI 2013, Harbin, China, June 12-15, 2013, Proceedings, Part I 4*, pp. 1–10. Springer, 2013.

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

[84] Lee, C., Kim, M., Kim, Y. J., Hong, N., Ryu, S., Kim, H. J., and Kim, S. Soft robot review. *International Journal of Control, Automation and Systems*, Vol. 15, pp. 3–15, 2017.

[85] 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.

[86] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[87] Lillicrap, T. P., Santoro, A., Marris, L., Akerman, C. J., and Hinton, G. Backpropagation and the brain. *Nature Reviews Neuroscience*, Vol. 21, No. 6, pp. 335–346, 2020.

[88] Liu, C., Chen, Y., Zhang, J., and Chen, Q. Cpg driven locomotion control of quadruped robot. In *2009 IEEE International Conference on Systems, Man and Cybernetics*, pp.

2368–2373. IEEE, 2009.

[89] Lorenz, E. N. Deterministic nonperiodic flow. *Journal of atmospheric sciences*, Vol. 20, No. 2, pp. 130–141, 1963.

[90] Lowe, R., Wu, Y. I., Tamar, A., Harb, J., Pieter Abbeel, O., and Mordatch, I. Multi-agent actor-critic for mixed cooperative-competitive environments. *Advances in neural information processing systems*, Vol. 30, 2017.

[91] Ma, K. Y., Chirarattananon, P., Fuller, S. B., and Wood, R. J. Controlled flight of a biologically inspired, insect-scale robot. *Science*, Vol. 340, No. 6132, pp. 603–607, 2013.

[92] Malley, M., Haghighat, B., Houe, L., and Nagpal, R. Eciton robotica: Design and algorithms for an adaptive self-assembling soft robot collective. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4565–4571. IEEE, 2020.

[93] McClelland, J. L., Rumelhart, D. E., Group, P. R., et al. *Parallel Distributed Processing, Volume 2: Explorations in the Microstructure of Cognition: Psychological and Biological Models*, Vol. 2. MIT press, 1987.

[94] McCulloch, W. S. and Pitts, W. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, Vol. 5, No. 4, pp. 115–133, 1943.

[95] McGuire, K., De Wagter, C., Tuyls, K., Kappen, H., and de Croon, G. C. Minimal navigation solution for a swarm of tiny flying robots to explore an unknown environment. *Science Robotics*, Vol. 4, No. 35, p. eaaw9710, 2019.

[96] McLurkin, J. and Yamins, D. Dynamic task assignment in robot swarms. In *Robotics: Science and Systems*, Vol. 8. Cambridge, USA, 2005.

[97] Miki, T., Lee, J., Hwangbo, J., Wellhausen, L., Koltun, V., and Hutter, M. Learning robust perceptive locomotion for quadrupedal robots in the wild. *Science Robotics*, Vol. 7, No. 62, p. eabk2822, 2022.

[98] Mirjalili, S., Mirjalili, S. M., and Lewis, A. Grey wolf optimizer. *Advances in engineering software*, Vol. 69, pp. 46–61, 2014.

[99] Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pp. 1928–1937. 2016.

[100] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[101] 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.

[102] 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.

[103] Morimoto, D., Hiraga, M., Shiozaki, N., Ohkura, K., and Munetomo, M. Generating collective behavior of a multi-legged robotic swarm using an evolutionary robotics approach. *Artificial Life and Robotics*, Vol. 27, No. 4, pp. 751–760, 2022.

[104] Mouret, J.-B. and Doncieux, S. Incremental evolution of animats' behaviors as a multi-objective optimization. In *International Conference on Simulation of Adaptive Behavior*, pp. 210–219. Springer, 2008.

[105] Nair, A., Srinivasan, P., Blackwell, S., Alcicek, C., Fearon, R., De Maria, A., Panneershelvam, V., Suleyman, M., Beattie, C., Petersen, S., et al. Massively parallel methods for deep reinforcement learning. *arXiv preprint arXiv:1507.04296*, 2015.

[106] Nakada, K., Asai, T., and Amemiya, Y. Biologically-inspired locomotion controller for a quadruped walking robot: Analog ic implementation of. *Journal of Robotics and Mechatronics*, Vol. 16, No. 4, p. 397, 2004.

[107] Naya, K., Kutsuzawa, K., Owaki, D., and Hayashibe, M. Spiking neural network discovers energy-efficient hexapod motion in deep reinforcement learning. *IEEE Access*, Vol. 9, pp. 150345–150354, 2021.

[108] Nolfi, S. Evolutionary robotics: Exploiting the full power of self-organization. 1998.

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

[110] Norris, J. R. *Markov chains*. 2. Cambridge university press, 1998.

[111] Nouyan, S., Campo, A., and Dorigo, M. Path formation in a robot swarm. *Swarm Intelligence*, Vol. 2, No. 1, pp. 1–23, 2008.

[112] Ohira, M., Chatterjee, R., Kamegawa, T., and Matsuno, F. Development of three-legged modular robots and demonstration of collaborative task execution. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pp. 3895–3900. IEEE, 2007.

[113] Okubo, A. Dynamical aspects of animal grouping: swarms, schools, flocks, and herds. *Advances in biophysics*, Vol. 22, pp. 1–94, 1986.

[114] Ozkan-Aydin, Y. and Goldman, D. I. Self-reconfigurable multilegged robot swarms collectively accomplish challenging terradynamic tasks. *Science Robotics*, Vol. 6, No. 56, p. eabf1628, 2021.

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

[116] Pollack, J. B. and Lipson, H. The golem project: Evolving hardware bodies and brains. In *Proceedings. The Second NASA/DoD Workshop on Evolvable Hardware*, pp. 37–42. IEEE, 2000.

[117] Price, K., Storn, R. M., and Lampinen, J. A. *Differential Evolution: A Practical*

Approach to Global Optimization. Natural Computing Series. Springer, 2005.

[118] Qin, B., Gao, Y., and Bai, Y. Sim-to-real: Six-legged robot control with deep reinforcement learning and curriculum learning. In *2019 4th International Conference on Robotics and Automation Engineering (ICRAE)*, pp. 1–5. IEEE, 2019.

[119] Reyes, P. and Escobar, M.-J. Neuroevolutive algorithms for learning gaits in legged robots. *IEEE Access*, Vol. 7, pp. 142406–142420, 2019.

[120] Roderick, W. R., Cutkosky, M. R., and Lentink, D. Bird-inspired dynamic grasping and perching in arboreal environments. *Science Robotics*, Vol. 6, No. 61, p. eabj7562, 2021.

[121] Romanishin, J. W., Gilpin, K., and Rus, D. M-blocks: Momentum-driven, magnetic modular robots. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 4288–4295. IEEE, 2013.

[122] Rubenstein, M., Ahler, C., and Nagpal, R. Kilobot: A low cost scalable robot system for collective behaviors. In *2012 IEEE international conference on robotics and automation*, pp. 3293–3298. IEEE, 2012.

[123] Şahin, E. Swarm robotics: From sources of inspiration to domains of application. In *Swarm Robotics: SAB 2004 International Workshop, Santa Monica, CA, USA, July 17, 2004, Revised Selected Papers 1*, pp. 10–20. Springer, 2005.

[124] Sahin, E., Labella, T. H., Trianni, V., Deneubourg, J.-L., Rasse, P., Floreano, D., Gambardella, L., Mondada, F., Nolfi, S., and Dorigo, M. Swarm-bot: Pattern formation in a swarm of self-assembling mobile robots. In *IEEE International Conference on Systems, Man and Cybernetics*, Vol. 4, pp. 6–pp. IEEE, 2002.

[125] Saputra, A. A., Takesue, N., Wada, K., Ijspeert, A. J., and Kubota, N. Aquro: A cat-like adaptive quadruped robot with novel bio-inspired capabilities. *Frontiers in Robotics and AI*, Vol. 8, p. 562524, 2021.

[126] Sarikaya, M. Biomimetics: materials fabrication through biology. *Proceedings of the National Academy of Sciences*, Vol. 96, No. 25, pp. 14183–14185, 1999.

[127] Schaul, T., Quan, J., Antonoglou, I., and Silver, D. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

[128] Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. Trust region policy optimization. In *International Conference on Machine Learning*, pp. 1889–1897. 2015.

[129] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

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

[131] Shepherd, R. F., Ilievski, F., Choi, W., Morin, S. A., Stokes, A. A., Mazzeo, A. D., Chen, X., Wang, M., and Whitesides, G. M. Multigait soft robot. *Proceedings of the national academy of sciences*, Vol. 108, No. 51, pp. 20400–20403, 2011.

[132] Shucker, B. and Bennett, J. K. Scalable control of distributed robotic macrosensors.

In *Distributed Autonomous Robotic Systems 6*, pp. 379–388. Springer, 2007.

[133] Siciliano, B. Kinematic control of redundant robot manipulators: A tutorial. *Journal of intelligent and robotic systems*, Vol. 3, pp. 201–212, 1990.

[134] 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.

[135] Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., and Riedmiller, M. Deterministic policy gradient algorithms. In *International conference on machine learning*, pp. 387–395. Pmlr, 2014.

[136] Sneyd, J., Theraula, G., Bonabeau, E., Deneubourg, J.-L., and Franks, N. R. *Self-organization in biological systems*. Princeton university press, 2001.

[137] 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.

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

[139] Sterman, J. D. Learning in and about complex systems. *System dynamics review*, Vol. 10, No. 2-3, pp. 291–330, 1994.

[140] 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.

[141] Strobel, V., Castelló Ferrer, E., and Dorigo, M. Managing byzantine robots via blockchain technology in a swarm robotics collective decision making scenario. 2018.

[142] 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.

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

[144] Sutton, R. S. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Advances in neural information processing systems*, pp. 1038–1044. 1996.

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

[146] Sutton, R. S., McAllester, D., Singh, S., and Mansour, Y. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, Vol. 12, 1999.

[147] Takahama, T. and Sakai, S. Constrained optimization by α constrained genetic algorithm (αga). *Systems and Computers in Japan*, Vol. 35, No. 5, pp. 11–22, 2004.

[148] Téllez, R. A., Angulo, C., and Pardo, D. E. Evolving the walking behaviour of a 12 dof quadruped using a distributed neural architecture. In *International Workshop on Biologically Inspired Approaches to Advanced Information Technology*, pp. 5–19.

Springer, 2006.

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

[150] Turgut, A. E., Çelikkanat, H., Gökçe, F., and Şahin, E. Self-organized flocking in mobile robot swarms. *Swarm Intelligence*, Vol. 2, No. 2, pp. 97–120, 2008.

[151] 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.

[152] Valsalam, V. K., Hiller, J., MacCurdy, R., Lipson, H., and Miikkulainen, R. Constructing controllers for physical multilegged robots using the enso neuroevolution approach. *Evolutionary Intelligence*, Vol. 5, No. 1, pp. 45–56, 2012.

[153] Van Hasselt, H., Guez, A., and Silver, D. Deep reinforcement learning with double q-learning. In *AAAI*, Vol. 2, p. 5. Phoenix, AZ, 2016.

[154] Vásárhelyi, G., Virágh, C., Somorjai, G., Nepusz, T., Eiben, A. E., and Vicsek, T. Optimized flocking of autonomous drones in confined environments. *Science Robotics*, Vol. 3, No. 20, p. eaat3536, 2018.

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

[156] Wang, Z., Schaul, T., Hessel, M., Van Hasselt, H., Lanctot, M., and De Freitas, N. Dueling network architectures for deep reinforcement learning. *arXiv preprint arXiv:1511.06581*, 2015.

[157] Watkins, C. J. C. H. *Learning from delayed rewards*. Ph.D. thesis, King's College, Cambridge, 1989.

[158] Williams, R. J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Reinforcement learning*, pp. 5–32, 1992.

[159] Wright, C., Johnson, A., Peck, A., McCord, Z., Naaktgeboren, A., Gianfortoni, P., Gonzalez-Rivero, M., Hatton, R., and Choset, H. Design of a modular snake robot. In *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2609–2614. IEEE, 2007.

[160] Yang, X.-S. and Hossein Gandomi, A. Bat algorithm: a novel approach for global engineering optimization. *Engineering computations*, Vol. 29, No. 5, pp. 464–483, 2012.

[161] 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.

[162] Ye, C., Ma, S., Li, B., and Wang, Y. Locomotion control of a novel snake-like robot. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems*

(IROS)(IEEE Cat. No. 04CH37566), Vol. 1, pp. 925–930. IEEE, 2004.

[163] Zahadat, P. and Schmickl, T. Division of labor in a swarm of autonomous underwater robots by improved partitioning social inhibition. *Adaptive Behavior*, Vol. 24, No. 2, pp. 87–101, 2016.

Appendix A

Supplemental Parts in Fitness Functions and Reward Function

This appendix describes supplemental parts ($\delta_1 \sim \delta_5$) used in fitness functions (Chapter 4 to Chapter 6) and reward functions (Chapter 7) to make the robot gait similar to legged animals. Both methods (i.e., evolutionary robotics and reinforcement learning) employed almost similar forms. The difference came from the frequency of calculation; fitness was calculated with relatively long-term results of robots, while reward was calculated in each timestep.

A.1 Fitness Function (Chapter 4 to Chapter 6)

The $\boldsymbol{\phi}_i$ in (4.3) of Chapter 4 is represented by follows:

$$\boldsymbol{\phi}_i = (\phi_{i,1},\ \ldots,\ \phi_{i,j},\ \ldots,\ \phi_{i,N_J}) \tag{A.1}$$

$$\phi_{i,j} = \sum_{t=1}^{T} \frac{|\theta_{i,j,t} - \theta_{i,j,t-1}|}{\theta_{\{i,j\}_{max}} - \theta_{\{i,j\}_{min}}}$$

where N_J is the number of joints equipped in the robot. The T is the number of simulation timesteps in each generation. The $\theta_{i,j,t}$[rad] is the angle of joint j in the robot i at timestep t. The $\theta_{\{i,j\}_{max}}$ and $\theta_{\{i,j\}_{min}}(\theta_{\{i,j\}_{max}} > \theta_{\{i,j\}_{min}})$ are maximum and minimum angles in a movable range of joint j.

For δ_3, χ_{i,δ_3} in (4.4) of Chapter 4 is the amount of change in the roll and pitch angles from baseline values. The χ_{i,δ_3} is calculated by the following equation:

$$\chi_{i,\delta_3} = \sum_{t=1}^{T}(|b_{roll} - \theta_{roll,i,t}| + |b_{pitch} - \theta_{pitch,i,t}|) \tag{A.2}$$

where b_{roll} and b_{pitch} are standard values for roll and pitch angles of the robot. The $\theta_{roll,i,t}$ and $\theta_{pitch,i,t}$ are roll and pitch angles of robot i at timestep t. In this paper, $b_{roll} = b_{pitch} = 0$. Therefore, robots are required to keep their posture parallel to the ground.

For δ_4, χ_{i,δ_4} is the amount of change in the bottom IR sensor from the standard value. The χ_{i,δ_4} is calculated by the following equation:

$$\chi_{i,\delta_4} = \sum_{t=1}^{T} |b_{IRb} - I_{IRb,i,t}| \tag{A.3}$$

where b_{IRb} is a standard value for the bottom IR sensor of the robot, and $I_{IRb,i,t}$ is the sensor value from the bottom IR sensor of the robot i at timestep t. In this paper, b_{IRb} is set as 0.5224 to make the distance between the floor and the body of robots around 0.1 m.

For δ_5, χ_{i,δ_5} is based on the number of times the direction of the rotating joint has changed. The χ_{i,δ_5} is calculated by the following equation:

$$\chi_{i,\delta_5} = \frac{1}{N_J} \sum_{j=1}^{N_J} \sum_{t=1}^{T} f_s(\dot{\theta}_{i,j,t}, \dot{\theta}_{i,j,t-1}) \tag{A.4}$$

$$f_s(x, y) = \begin{cases} 1 & \text{if sign}(x) \neq \text{sign}(y) \\ 0 & \text{otherwise} \end{cases}$$

where $\dot{\theta}_{i,j,t}$, is the angular velocity of joint j in the robot i at timestep t. The sign(x) returns whether an argument x is positive or negative.

A.2 Reward Function (Chapter 7)

In Eq. 7.5, δ_1 to δ_5 are supplemental parts to make the robot's gait similar to natural organisms. The δ_1 to δ_5 are designed as in Chapter 4 (or [103]) to avoid unnatural gaits, for instance, shaking the torso frequently or using only a few legs. The δ_1 is for walking forward, which is represented by the following equation:

$$\delta_1 = K_{\delta_1} \cdot \sum_{t}^{t+T_{r_1}-1} \max\left(\frac{\boldsymbol{f} \cdot \boldsymbol{v}_{t-T_{r_1}+1}}{|\boldsymbol{f}||\boldsymbol{v}_{t-T_{r_1}+1}|}, 0 \right) \tag{A.5}$$

where $\boldsymbol{v}_{t\leq 0} = \boldsymbol{0}$. The K_{δ_1} is a constant value, $\boldsymbol{v}_t$ is the velocity vector of a robot, and $\boldsymbol{f} = (0, 1, 0)$ is the vector indicating a front direction on the local coordinate of robots. The T_{r_1} is the number of latent timesteps for calculating r_1. δ_1 is the function of cosine similarity for the $\boldsymbol{v}_t$ and the $\boldsymbol{f}$. The value of δ_1 increases when a robot shows walking forward.

The δ_2 is for driving joints equally, which is represented as follows:

$$\delta_2 = K_{\delta_2} \cdot \frac{|\boldsymbol{\phi}|}{|\boldsymbol{\phi}| + c} \cdot \left(\frac{\boldsymbol{\phi} \cdot \boldsymbol{1}}{|\boldsymbol{\phi}||\boldsymbol{1}|} \right)^{\beta} \tag{A.6}$$

where K_{δ_2}, c, and β are constant values, and $\boldsymbol{1}$ is a vector whose elements are all 1. The $\boldsymbol{\phi}$ is the vector indicating the rotation angles of each joint. The $\boldsymbol{\phi}$ is represented by follows:

$$\begin{aligned} \boldsymbol{\phi} &= (\phi_1, \ \ldots, \ \phi_i, \ \ldots, \ \phi_{N_J}) \\ \phi_i &= \sum_{t}^{t+T_{r_1}-1} \frac{|\theta_{i,t-T_{r_1}+1} - \theta_{i,t-T_{r_1}}|}{\theta_{i,\max} - \theta_{i,\min}} \end{aligned} \tag{A.7}$$

where $\theta_{i,t\leq 0} = 0$. The N_J is the number of joints equipped in the robot. The $\theta_{i,t}$[rad] is the angle of joint i in a robot at timestep t. The $\theta_{i,\max}$ and $\theta_{i,\min}(\theta_{i,\max} > \theta_{i,\min})$ are maximum and minimum angles in a movable range of joint i. The ϕ_i showed the rotated amount of joint j in recent T_{r_1} timesteps. The value of δ_2 increases when robots move each leg equally.

The δ_3, δ_4, and δ_5 are represented by the following equations:

$$\delta_j = \min\left(\frac{c'_{\delta_j}}{1 + c''_{\delta_j}\chi_{\delta_j}^{\gamma_{\delta_j}}}, 1\right) \cdot a_{\delta_j} + b_{\delta_j}, \tag{A.8}$$

where $j \in \{3, 4, 5\}$. The c'_{δ_j}, c''_{δ_j}, γ_{δ_j}, a_{δ_j}, and b_{δ_j} are constant values. The χ_{δ_j} is the scalar value that shows the robot state to be minimized. For δ_3, χ_{δ_3} is the amount of change in the roll and pitch angles from standard values. The χ_{δ_3} is calculated by the following equation:

$$\chi_{\delta_3} = \sum_{t}^{t+T_{r_1}-1} (|b_{\text{roll}} - \theta_{\text{roll},t-T_{r_1}+1}| + |b_{\text{pitch}} - \theta_{\text{pitch},t-T_{r_1}+1}|) \tag{A.9}$$

where $\theta_{\text{roll},t\leq 0} = b_{\text{roll}}$, and $\theta_{\text{pitch},t\leq 0} = b_{\text{pitch}}$. The b_{roll} and b_{pitch} are the standard values for the roll and pitch angles of the robot. The $\theta_{\text{roll},t}$ and $\theta_{\text{pitch},t}$ are the roll and pitch angles of a robot at timestep t. The χ_{δ_3} counts the changes of roll and pitch angles from b_{roll} and b_{pitch} in recent T_{r_1} timesteps. In this paper, $b_{\text{roll}} = b_{\text{pitch}} = 0$. Therefore, robots are required to keep their posture parallel to the ground.

For δ_4, χ_{δ_4} is the amount of change in the bottom proximity sensor from the standard value. The χ_{δ_4} is calculated by the following equation:

$$\chi_{\delta_4} = \sum_{t}^{t+T_{r_1}-1} |b_{\text{pb}} - I_{\text{pb},t-T_{r_1}+1}| \tag{A.10}$$

where $I_{\text{pb},t\leq 0} = b_{\text{pb}}$. The b_{pb} is a standard value for the bottom proximity sensor of the robot, and $I_{\text{pb},t}$ is the sensor value from the bottom proximity sensor of a robot at timestep t. In this paper, $I_{\text{pb},t}$ is set as 0.5224 to make the proximity between the floor and the body of robots around 0.1 m.

For δ_5, χ_{δ_5} is based on the number of times the direction of the rotating joint has changed. The χ_{δ_5} is calculated by the following equation:

$$\chi_{\delta_5} = \frac{1}{N_J}\sum_{i=1}^{N_J}\sum_{t}^{t+T_{r_1}-1} f_s(\dot{\theta}_{j,t-T_{r_1}+1}, \dot{\theta}_{j,t-T_{r_1}}) \tag{A.11}$$

$$f_s(x, y) = \begin{cases} 1 & \text{if sign}(x) \neq \text{sign}(y) \\ 0 & \text{otherwise} \end{cases}$$

where $\dot{\theta}_{i,t\leq 0} = 0$. The $\dot{\theta}_{i,t}$ is the angular velocity of joint i in a robot at timestep t. The sign(x) returns whether an argument x is positive or negative. From δ_1 to δ_5 are available when $t > T_{r_1}$.

Appendix B

Parameter Settings

B.1 Chapter 4

The parameter values used in Chapter 4 are summarized in Table B.1. Most parameters are shared in Chapter 5 and Chapter 6 except for K_1 and K_2.

Table B.1. Parameter settings in Exp-0 and Exp-1 of Chapter 4. The bold font shows the changed parameters in Exp-1.

Parameter	Exp-0	Exp-1	Parameter	Exp-0	Exp-1
T	5000 (166[s])	5000	c_{4,δ_4}	300.0	300.0
N_r	1	**10**	c_{5,δ_4}	2.0×10^{-3}	2.0×10^{-3}
c_1	1.0	1.0	γ_{δ_4}	2.0	2.0
c_2	1.0×10^{-3}	**0.01**	a_{δ_4}	0.66	0.66
$\boldsymbol{x_t}$	(1000, 0, 0)	(1000, 0, 0)	b_{δ_4}	0.94	0.94
α	0.5	0.5	c_{4,δ_5}	4.0×10^{11}	4.0×10^{11}
S	80.0	80.0	c_{5,δ_5}	8.0×10^{-8}	8.0×10^{-8}
K_{δ_1}	1.0	1.0	γ_{δ_5}	6.0	6.0
K_{δ_2}	1.0	1.0	a_{δ_5}	1.1	1.1
c_3	200.0	200.0	b_{δ_5}	0.9	0.9
β	4.0	4.0	ρ	0.6	0.6
c_{4,δ_3}	10.0	10.0	I	-	**40.0**
c_{5,δ_3}	1.5×10^{-4}	1.5×10^{-4}	M	1	1
γ_{δ_3}	2.0	2.0	K_1	3190.7	3190.7
a_{δ_3}	0.7	0.7	K_2	0.0	$\mathbf{4.0 \times 10^{-4}}$
b_{δ_3}	0.9	0.9			

Table B.2. Parameter settings of PPO in Chapter 7.

Parameter	Value
Horizon (T)	128
Adam stepsize	3.0×10^{-4}
Num. epochs	10
Minibatch size	64×16
Discount (γ)	0.995
GAE parameter (λ)	0.97
Number of actors	16
Clipping ϵ	0.05
Learning steps	1.6×10^{7}

Table B.3. Parameters for reward settings in Chapter 7.

Parameter	Value	Parameter	Value
K_1	1.0	γ_{δ_3}	2
a	1.25	a_{δ_3}	0.7
$v_{\max}$	0.024	b_{δ_3}	0.9
K_2	0.3	c'_{δ_4}	300.0
$E_{r_{\text{change}}}$	50	c''_{δ_4}	0.2
α	0.04	γ_{δ_4}	2
K_{δ_1}	0.02	a_{δ_4}	0.66
T_{r_1}	50	b_{δ_4}	0.94
K_{δ_2}	1.0	c'_{δ_5}	4.0×10^{11}
c	2.0	c''_{δ_5}	8.0×10^{-6}
β	4	γ_{δ_5}	6
c'_{δ_3}	10.0	a_{δ_5}	1.1
c''_{δ_3}	1.5×10^{-2}	b_{δ_5}	0.9

B.2 Chapter 7

The parameter setting of PPO is summarized in Table. B.2. The parameters for the reward setting are summarized in Table. B.3.

Appendix C

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 D.

Chapter 3

- 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*, Vol. 27, No. 2, pp. 333–340, 2022.
- 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.

Chapter 4

- Daichi Morimoto, Motoaki Hiraga, Naoya Shiozaki, Kazuhiro Ohkura, and Masaharu Munetomo. Generating collective behavior of a multi-legged robotic swarm using an evolutionary robotics approach. *JArtificial Life and Robotics*, Vol. 27, No. 4, pp. 751–760, 2022.
- 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.

Chapter 5

- Haruhi Tsukamoto, Daichi Morimoto, Motoaki Hiraga, Kazuhiro Ohkura, and Masaharu Munetomo. Evolving collective behavior of a multi-legged robotic swarm in a rough terrain environment. In *Proceedings of the 6th International Symposium on Swarm Behavior and Bio-Inspired Robotics*, pp. 1544–1549, 2023.

Chapter 6

- Daichi Morimoto, Motoaki Hiraga, Kazuhiro Ohkura, and Masaharu Munetomo. Generating and analyzing collective step-climbing behavior in a multi-legged robotic swarm. In *Proceedings of the Thirteenth International Conference on Swarm Intelligence*, pp. 324–331, 2022.

Chapter 7

- Daichi Morimoto, Yukiha Iwamoto, Motoaki Hiraga, and Kazuhiro Ohkura. Generating collective behavior of a multi-legged robotic swarm using deep reinforcement learning. *Journal of Robotics and Mechatronics*, accepted.

Appendix D

List of Publications

Journal Publications

1. Daichi Morimoto, Yukiha Iwamoto, Motoaki Hiraga, and Kazuhiro Ohkura. Generating collective behavior of a multi-legged robotic swarm using deep reinforcement learning. *Journal of Robotics and Mechatronics*, accepted.
2. Motoaki Hiraga, Daichi Morimoto, Yoshiaki Katada, and Kazuhiro Ohkura. When less is more in embodied evolution: Robotic swarms have better evolvability with constrained communication. *Journal of Robotics and Mechatronics*, accepted.
3. Daichi Morimoto, Motoaki Hiraga, Naoya Shiozaki, Kazuhiro Ohkura, and Masaharu Munetomo. Generating collective behavior of a multi-legged robotic swarm using an evolutionary robotics approach. *Artificial Life and Robotics*, Vol. 27, No. 4, pp. 751–760, 2022.
4. 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*, Vol. 27, No. 2, pp. 333–340, 2022.
5. 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.

International Conferences

1. Naoya Shiozaki, Yu Watanabe, Daichi Morimoto, Motoaki Hiraga, and Kazuhiro Ohkura. Attention agent for collective behavior of a robotic swarm with deep neuroevolution. In *Proceedings of the 6th International Symposium on Swarm Behavior and Bio-Inspired Robotics*, pp. 1550–1555, 2023.
2. Haruhi Tsukamoto, Daichi Morimoto, Motoaki Hiraga, Kazuhiro Ohkura, and Masaharu

Munetomo. Evolving collective behavior of a multi-legged robotic swarm in a rough terrain environment. In *Proceedings of the 6th International Symposium on Swarm Behavior and Bio-Inspired Robotics*, pp. 1544–1549, 2023.

3. Daichi Morimoto, Motoaki Hiraga, Kazuhiro Ohkura, and Masaharu Munetomo. Generating and Analyzing Collective Step-climbing Behavior in a Multi-legged Robotic Swarm. In *Proceedings of the Thirteenth International Conference on Swarm Intelligence*, pp. 324–331, 2022.
4. 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.
5. Gan Weng, Daichi Morimoto, Boyin Jin, and Kazuhiro Ohkura. Generating Collective Wall-jumping Behavior with a Robotic Swarm Using Deep Reinforcement Learning. In *Proceedings of the 5th International Symposium on Swarm Behavior and Bio-Inspired Robotics*, pp. 1776–1781, 2022.
6. 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.
7. 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.

Domestic Conferences

1. 宮本明治, 平賀元彰, 森本大智, 大倉和博. Surrogate-Assisted MBEANN の提案とその評価. 第67回システム制御情報学会研究発表講演会講演論文集, 115-6, pp. 113–118, 2023.
2. 小村真央, 森本大智, 平賀元彰, 大倉和博. 自己適応型突然変異法を導入した MBEANN の提案とその効果の検証. 第67回システム制御情報学会研究発表講演会講演論文集, 115-5, pp. 107–112, 2023.
3. 潮崎直哉, 森本大智, 平賀元彰, 大倉和博. Attention Agent 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. 第23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, 1P3-A18, pp. 1046–1051, 2022.
4. 塚本遙日, 森本大智, 平賀元彰, 大倉和博, 棟朝雅晴. Neuroevolution を用いた多脚自律ロボットスワームの不整地環境での群れ行動生成. 第23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, 1P3-A14, pp. 1031–1036, 2022.
5. 岩本透羽, 森本大智, 平賀元彰, 大倉和博. 深層強化学習による多脚自律ロボットスワームの

群れ行動生成. 第66回システム制御情報学会研究発表講演会講演論文集, 211-5, pp. 416–422, 2022.

6. 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.
7. 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.
8. 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.
9. 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.
10. 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.
11. 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.
12. 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.
13. 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.
14. 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.
15. 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.
16. 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.
17. 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.
18. 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.

19. 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.
20. 内田隼, 森本大智, 大倉和博. Random walk するロボティックスワームの特徴判別能力と確率分布の関係. ロボティクス・メカトロニクス講演会2019講演論文集, 1P2-H08, 2019.
21. 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.
22. 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.
23. 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.
24. 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.
25. 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.