2 Background and Related Work

This chapter comprises three parts. Section 2.1 briefly introduces the background, including the development of artificial intelligence (AI) and robotics, and the research that has been done in recent years in the area of automation science. Section 2.2 reviews the development of evolutionary computation, which is the main technique used in this thesis. This section includes an introduction of biological evolution, principles, strengths and weaknesses of evolutionary computation, and its applications. Section 2.3 reviews how AI/robotics and animal behavior study benefit from each other. It gives examples of how animal behavior can be used as inspiration for AI and robotics and of methods to investigate animal behavior using AI/robotics techniques.

2.1 Background

2.1.1 The Development of AI and Robotics

Intelligence is a natural part of life. Humans and other biological creatures exhibit many behaviors (e.g., decision making and foraging) that require intelligence. However, intelligence is not a property that is limited to biological creatures. It should be equally applicable to machines. The term AI emerged in a conference in 1956 at Dartmouth College, where several researchers (e.g., Marvin Minsky and John McCarthy) who were latter considered pioneers of this field, discussed the development of digital computers and the future of AI. The definition of AI is still a disputed topic. Some researchers argue that AI is to simulate the intelligent behaviors that are observed in humans and other biological creatures using computers or machines. That is, an intelligent machine should be able to exhibit behaviors similar to that of living creatures when encountering the same problems [23]. Others give the following definition: "Artificial Intelligence is

the study of mental faculties through the use of computational models" [24]. According to Fogel [25], an intelligent system should know how to make decisions in order to fulfill a goal (e.g., solving a problem). In other words, instead of pre-programming the machine using human's knowledge, it should be able to learn and adapt. In [26], Minsky argued, "Why can't we build, once and for all, machines that grow and improve themselves by learning from experience? Why can't we simply explain what we want, and then let our machines do experiments or read some books or go to school, the sorts of things that people do?" In 1950, Turing [27] proposed an imitation game which is nowadays known as *Turing test* to discuss a question: "Can machine think?". Although whether a machine could pass the *Turing test* or not is beyond the consideration at that time, it was accepted as a notion that a machine could learn and adapt to mimic human behavior. Many promising achievements have been made to enable machines to accomplish a variety of intelligent actions since then.

In the 1970s, the emergence of expert systems—computer programs that mimic human experts' decision-making capability [28], significantly promoted the development of AI. An expert system can solve complicated problems through reasoning about the knowledge (which is mainly represented as *if-then* rules) it has. In an expert system, there are two elements—knowledge base and inference engine. Knowledge base includes facts and rules that are known to the system. Inference engine utilizes the facts and rules to make decisions and derive new rules, which are then stored in the system to update the knowledge base. Expert systems have many real-world applications such as medical diagnosis [29], scientific hypothesis formation [30] and structural safety monitoring [31]. The rules in an expect system can be expressed using Boolean logic or Fuzzy logic. In Boolean logic, every condition in the rules is either true or false. Fuzzy logic was introduced by Zadeh [32] to describe "degree of truth". For example, a cup with water is described as "full (1)" or "empty (0)" using Boolean logic; however, in Fuzzy logic, it can also be described using some fuzzy expressions such as "almost full", "half full", "near empty". Fuzzy logic is commonly used in our daily life. An example rule in an expert system using Fuzzy logic is: *IF the temperature is cold, THEN turn the heater on.* In stock markets an old saying is: "buy low, sell high". However, whether the stock value can be considered as low or high depends on the stock curves in a particular situation. Fuzzy systems have many commercial applications, such as in air conditioners, digital cameras and hand writing recognition. Another representation of AI are neural networks, which mimic the processing ability of nervous systems of biological creatures (especially human brain). Through a combination of weights and excitation functions (e.g., sigmoid func-

tion), neural networks can accomplish many tasks observed in humans, such as pattern recognition and image processing.

Robotics is a field about making machines that can move in several ways to accomplish certain tasks. While AI and robotics are not essentially connected, they are often used together to make robots appear intelligent. For example, a robot with AI can move autonomously and make decisions while interacting with the environment it is operating in. Through combining AI and robotics, machines can be created to behave more like humans or animals which can learn and adapt to the changing environment. In [33], a robot was built to be capable of automatically finding compensatory behaviors after going through damage to accomplish a locomotion task.

There are two common paradigms adapted for the control of a robot: deliberative paradigm and reactive paradigm. In deliberative paradigm, the robot operates on a top-down fashion and its action mainly depends on planning. A typical cycle is: *sense* $\rightarrow$ *plan* $\rightarrow$ *act*. In the sensing stage, the robot gets the information from the world based on sensors such as cameras, infrared sensors, etc. After pre-processing, this information would be passed to the central control architecture which integrates all the sensing information and reasons about it. Based on the knowledge the robot has it can decide which action to take to fulfill a goal (e.g. maximize its reward). This paradigm has led to many successful applications. The pioneer work is *Shakey the robot*, which is capable of reasoning about its own actions [34]. In this work, the environment the robot is operating in is simplified and the experimental conditions are well controlled (e.g. uniform color and flat floor). More work has been done since then to enable robots to tackle complex and changing environments [35]. In the reactive paradigm, the robot makes decisions based on *sense* $\rightarrow$ *act* without deliberate reasoning or planing. Instead of building a central reasoning system to integrate all the sensory information, the robot could process it in parallel. Some famous robots using reactive paradigm are Allen [36], Herbert [37] and Genghis [38].

2.1.2 Introduction of Automation Science

With the development of AI and robotics, intelligent and automation systems were commonly used for assisting scientific research. Since the first clinical automated laboratory management system [39] was created in 1993, such systems are increasingly used in

drug discovery, agriculture and energy labs, etc. In order to accelerate the experimental process, researchers take advantage of intelligent and automation systems to help, for example, collect and analyze data, as this can be time-consuming and tedious if carried out manually. The ideal situation is to make a machine conduct scientific research automatically without or with little human intervention. It could then conduct experiments day and night in a constant manner without any tiredness.

[big gap for some reason

The field of automation science has been developed to a great extent because of the increasing demands of drug industry and relevant fields of biology and chemistry. High-throughput screening (HTS) systems [40] are one of the early efforts. HTS systems can do many things such as preparation, observation and data analysis, greatly enhancing the speed of experimental process. Recently, King, et al. [41, 7] have built a "Robot Scientist"—Adam. It can automatically generate functional genomics hypotheses about the yeast *Saccharomyces cerevisiae* and carry out experiments to test and refine the hypotheses based on techniques developed in AI and robotics. Adam could automatically conduct the cycles of scientific experiments: forming hypothesis, initializing the experiments, explaining the results and verifying the hypothesis, and then repeating the cycle. The functional genomics hypotheses are autonomously generated by intelligent software and the experiments are conducted coordinating different components in the automated system [41]. This "Robot Scientist" [7] is able to conduct plenty of experiments and observations a day. In [42], Gauld et al. have developed a digital automated identification system (DAISY) to identify biological species automatically with high accuracy using an advanced image processing technique. This technology has gone though great improvement in recent years, enhancing the possibility of automation, or at least semi-automation, in the process of routine taxonomic identification. In [43], MacLeod et al. reported that an imaging system that is originally designed for identifying marine zooplankton was used by the US government for automatically monitoring Deepwater Horizon oil spill. They argue that taxonomists and researchers in machine learning, pattern recognition as well as artificial intelligence should collaborate with each other in order to better identify and name biological species. Drawing on approaches from various research areas, especially AI and robotics, intelligent and automation systems are playing a vital role in scientific research, allowing researchers to conduct experiments more efficiently. It is argued that the revolution of automation science would emerge in a few decades [7].

the increasing demands of

the relevant fields

a hypothesis

by coordinating

doesn't sound scientific

in a day

have

was originally

the Deepwater Horizon oil spill

or oil spills ?

used this reference 3 times in this paragraph.

2.2 Evolutionary Computation

Evolutionary computation is a field that draws inspiration from biological evolution. We use an evolutionary computation technique to automate generation of models during the system identification process. This section is organized as follows. Section 2.2.1 introduces biological evolution. Section 2.2.2 details the principles, strengths and weaknesses of evolutionary computation. Section 2.2.3 presents three main applications of evolutionary computation and related work.

2.2.1 Biological Evolution

Biological evolution is about how living creatures evolve to adapt to their changing environment. According to Darwin's Theory of Evolution [44], individuals compete for survival. The individuals that are best adapted to the environment tend to reproduce. This phenomenon is regarded as *natural selection*. From another point of view, the genes that help the species survive would have a higher chance of being preserved and passed on to the next generation, while the genes that are harmful would be abandoned.

Natural selection tends to accumulate small beneficial genetic mutations [45]. Suppose that some members in a species have evolved a functional organism that is very useful (e.g., a wing that can fly). This makes these members easier to find food or avoid predators. Their offspring are more likely to inherit such advantageous function and this function would be passed to the next generation. The other members without the advantageous function are unlikely to reproduce. Natural selection helps the species to adapt better to their environment. At the same time, it also accelerates the extinction of the species that can not adapt well to the environment. The dinosaur used to be a dominant species in the past due to its big body. This was a big advantage when the climate was mild. However, as the climate changed dramatically (e.g., extremely cold or hot), the big body was no longer an advantage as it needed to consume much more energy. This accelerated the extinction of dinosaurs [46].

Coevolution is a special form of evolution, which involves the simultaneous evolution of two or more dependent species. A typical example of coevolution is fox and rabbit or parasite and host. In nature, the survival ability between species is coupled. That means the survival ability of one particular individual in a species depends not only

Table 2.1: The relationship between different species that coevolve. Symbols '+', '-' and '0' represent beneficial, harmful and neutral in a relationship, respectively. For example, "+, -" indicates A benefits and B get harmed.

	the influence of species A		
the influence of species B	+,+ (reciprocity)	+,0 (symbiosis)	+,- (predation)
	0,+ (symbiosis)	0,0 (neutral)	0,- (amensalism)
	-,+ (predation)	-,0 (amensalism)	-,-(antibiosis)

on its chromosome, but also the interaction with the individuals from other species. For a specific species, there can be three relationships with other species: beneficial, harmful and neutral. Therefore, through permutation and combination, the relationship between two species can be summarized in Table 2.1. It includes 6 concrete relationships: reciprocity, neutral, symbiosis, amensalism, predation and antibiosis [47].

2.2.2 Introduction of Evolutionary Computation

2.2.2.1 Principles

Based on the principle of biological evolution, the genetic algorithm (GA) was proposed by Holland in 1960s [48]. There is a population of solutions in GA. The solution for a given problem is represented as a chromosome, which contains several genes. Each gene could be a bit, integer, or floating-point number. The GA is driven by a fitness function, which defines the quality of the candidate solutions in the population. The evolutionary process is to optimize (e.g., maximize) the fitness of the individuals. There are three genetic operators: selection, crossover and mutation. In each generation, the individuals with high fitness have a higher chance of being selected to the next generation and producing offsprings (e.g., through crossover). Mutation is normally realized by randomly changing a particular gene. For example, some gene in the chromosome may be randomly replaced by another gene. Mutation in GA serves the same function as it is in biological evolution. It creates diversity in the population.

There are other types of evolutionary algorithms. Fogel, Owens and Walsh invented *evolutionary programming (EP)* [49]. Rechenberg and Schwefel introduced *evolution*

strategies (ES) [50, 51], which are mainly dealing with real-value continuous optimization problems. In the early 1990s, another type of evolutionary algorithm called *genetic programming (GP)* was presented by Koza [52]. Fig. 2.1 shows a brief classification of evolutionary algorithms.

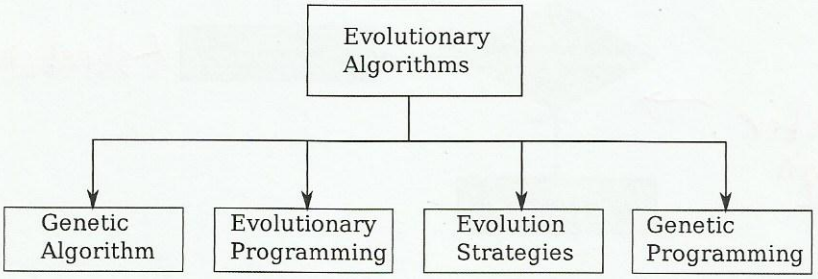


Figure 2.1: Classification of evolutionary algorithms

The implementation of evolutionary algorithms follows a general flow during the operation process. They can be divided into five steps: initialization, evaluation, mutation, selection and termination. At the beginning, a random population of individuals is initialized. Each individual is represented as a string (chromosome), which contains several genes (e.g., floating point numbers). These strings encode the candidate solutions. Different chromosomes are then evaluated using a predefined fitness function. The fitness of individuals in the population only depends on their own chromosomes. Selection happens after evaluation of each individual, and the ones with higher fitness are more likely to be selected to the next generation and have offsprings. The offsprings would go through mutation, which helps to maintain the diversity of the whole population. Fig. 2.2 show a diagram of how the evolutionary algorithms proceed.

As we mentioned before, the creatures in nature are not independent, and they have different kinds of relationship as shown in Table 2.1. The evolution of individuals among different species are coupled. Coevolutionary algorithms, which coevolve simultaneously two or more populations, are also widely adapted to solve real-world problems [54, 55, 56, 57]. In principle, coevolutionary algorithms can be considered of comprising several sub-algorithms, each of which could be an evolutionary algorithm. These sub-algorithms interact with each other in the fitness calculation process. In other words, the fitness of individuals in one population not only depends on its own chromosome, but also on the performance of other individuals from another population during the coevolutionary process.

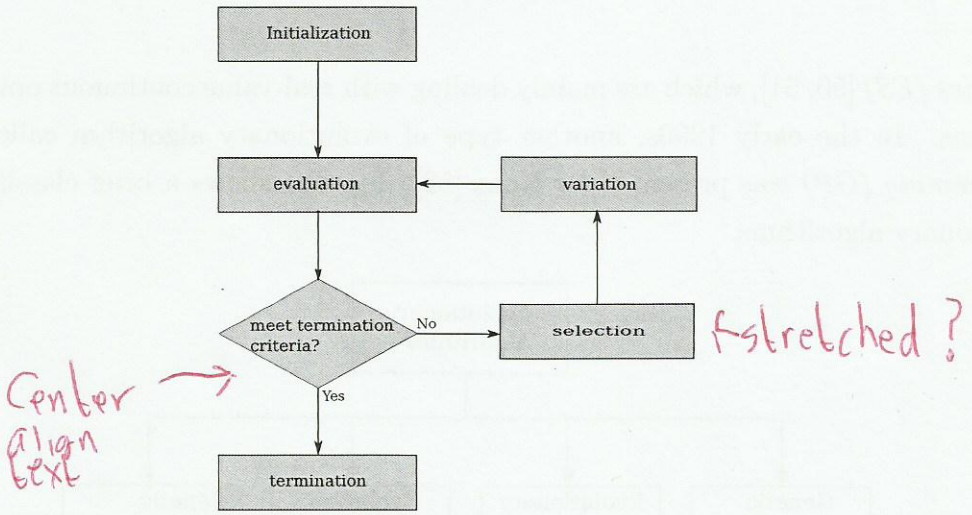


Figure 2.2: This diagram shows the flow of evolutionary algorithms. Adapted from [53]

The essential difference between evolutionary algorithms and coevolutionary algorithms is the way of fitness evaluation. In coevolutionary algorithms, the individual's fitness depends on its performance and that of the individuals from the other populations. Coevolutionary algorithms are generally divided into two categories—competitive coevolutionary algorithms (Comp-CEA) and cooperative coevolutionary algorithms (Coop-CEA). A Comp-CEA assesses each individual by its competitive performance with respect to its opponents, while Coop-CEA assesses each individual by its cooperative performance with respect to its co-operators. As discussed by Dawkins and Krebs [58], competitive coevolution can produce the phenomena of "arm races" where complexity of each population increases. The evolution of one population may drive another population to evolve new strategies, which makes the individuals of both populations evolve a higher level of complex behavior. Generally, Coop-CEA is applied to the situation in which the problem can be divided into several sub-problems. In Coop-CEA, there are several cooperative populations evolving simultaneously, and each individual of a sub-population represents a part of the solution. The fitness of an individual could be the quality of the solution formed by the combination of the individual with those from other sub-populations. In the rest of the thesis, we only discuss Comp-CEAs, which will also be referred to as coevolutionary algorithms in general.

In coevolutionary algorithms, the fitness of individuals is called subjective fitness [59]. An individual's subjective fitness is based on the performance of its temporary opponents from the current generation or a combination of current and past generations. The fitness

of individuals with the same chromosome may vary in different generations because of changing opponents. Conversely, the fitness in evolutionary algorithms is called absolute or objective fitness.

Suppose there are two populations in a coevolutionary algorithm. One is called "learner", and the other is called "evaluator". Let L represent a set of learners and E represent a set of evaluators. For a learner, a simple way of calculating its subjective fitness is to count the number of evaluators (in the current population) that this learner defeated [60]. It is described in equation (2.1) as follows (reproduced from [47]):

$$\forall i \in L \Rightarrow CF_i = \sum_{j \in E, i\, defeats\, j} 1. \tag{2.1}$$

CF_i is the fitness of learner i.

Another fitness calculation approach is called competitive fitness sharing [54] as described in the following (reproduced from [47]):

$$\forall j \in E \Rightarrow N_j = \sum_{k \in L, k\, defeats\, j} 1. \tag{2.2}$$

$$\forall i \in L \Rightarrow CF_i = \sum_{j \in E, i\, defeats\, j} \frac{1}{N_j}. \tag{2.3}$$

N_j represents the number of learners that could defeat evaluator j.

In competitive fitness sharing [54], the learner that could defeat the more competitive evaluator gets higher reward. For example, if a learner i, in a population is the only individual to defeat an evaluator j, this learner's accumulative fitness is added by 1, as N_j is equal to 1 in Equation (2.3). The aim of using competitive fitness sharing is to preserve/award the learner that possesses genetic material that is worth passing to the next generation.

There are many ways to choose the evaluators (temporary opponents). Random Pairing [61] means finding a random temporary opponent for each learner. In single elimination tournament [62], all the individuals randomly matched and the losers are taken out and

the winners are selected into next the round of random matches. Round Robin [61] means all the evaluators are the temporary opponents of each learner. There are also other ways such as K-random opponent [62] and shared fitness [54]. The evaluation time for random pairing is the shortest, but the performance is the worst; the calculation time for round robin is the longest, but the performance is the best [47]. In our coevolutionary algorithm, we chose to use the fitness calculation similar to Equation (2.1) and Round Robin.

2.2.2.2 Strengths

The advantages of evolutionary computation can be summarized in the following [63]:

- *Conceptually Simple:* Evolutionary computation techniques can be implemented using simple genetic operators (e.g., selection, crossover, mutation) as mentioned in Section 2.2.2.1.
- *Task-independent:* Evolutionary computation techniques can be used for optimization. Many tasks in reality can be treated as function optimization or black-box optimization problems (which will be detailed in Section 2.2.3.1).
- *Parallel Computing:* Evolutionary algorithms can be run in parallel to accelerate the evolutionary process. The individual can be evaluated independently according to the fitness function. However, the selection and recombination process is normally done in series.
- *Robust to Changes:* Evolutionary computation is robust to changes in circumstances. Once the circumstances have changed, the evolved solutions can be used as a start for further development without the need to restart the whole process [64].

Apart from the general advantages of evolutionary computation, coevolutionary algorithms have the following two advantages:

- *Open-Ended Evolution:* Coevolutionary algorithms can create an open-ended evolution for each population due to the complex interaction between the competing populations during the coevolutionary process. Such open-ended evolution could

encourage the appearance of new building blocks, thus maintaining the diversity of populations. In Darwin's natural selection, this phenomenon is referred to as "arm race" [58], which leads each species to continuously improve.

- *No Absolute Fitness Needed:* Coevolutionary algorithms can be applied to solve problems in which absolute fitness can not be effectively defined. For example, when evolving a chess program, it is challenging to define a fitness to qualify it, although playing against fixed programs would be an option [65]. An alternative way of evaluating a chess program is making it play with competing programs and then calculating its subjective fitness [60, 66].

2.2.2.3 Weaknesses

The main weakness of evolutionary computation is that it usually requires significant (computational) efforts to obtain good solutions. In simulation, this would not necessarily be a problem; however when the evaluation needs to be conducted on a physical system, it would take a long time and may also be expensive.

There is no guarantee that using evolutionary computation techniques can always find the 'perfect' or right solution. This may limit its use on some tasks that have a high demand of safety, as an occasional failure may cause the system to crash.

Apart from the general disadvantages of evolutionary computation techniques, in particular, there are three main pathologies of coevolutionary algorithms:

- *Red Queen Effect:* When Red Queen effect happens, two populations compete with each other and their subjective fitness keeps increasing during the coevolutionary process, however their objective fitness does not make constant improvement. On the other hand, their objective fitness increases, but the subjective fitness does not reflect such situation [55].

- *Cycling:* In coevolutionary algorithms, the criteria (temporary opponents) used for evaluating an individual (solution) is changing over generations. As a consequence of this, some good solutions in the previous generations may be lost and rediscovered later. This phenomenon is referred to as *cycling* [59]. A method of overcoming the problem of "cycling" is "hall of fame"[54]. "Hall of fame" remains

the good individuals from the previous generations, and these obtained individuals may be selected to be temporary components to evaluate the individuals from the competing population in the current generation.

- *Disengagement:* During the coevolutionary process, when one population is entirely better than the other, disengagement will occur. In this case, the selection criteria will not make sense, since the subjective fitness of each population is constant. This would cause each population to drift and fail to find the right solutions. Common methods for addressing the disengagement problem is "resource sharing" [67] and "reducing virulence" [59].

2.2.3 Applications of Evolutionary Computation

Evolutionary computation techniques are widely used for solving various engineering tasks such as image processing, pattern recognition, robot control and system identification. Many real-world applications of evolutionary computation can be found. In the area of nanophotonic light trapping, a need is the development of low cost thin film solar photovoltaic technologies. A traditional way is fixing the structure according to the physical intuition and trying to find the good parameters. In [68], a highly efficient light-trapping structure was designed using genetic algorithm. It was shown that this structure can increase the trapping efficiency three times compared with the classic design. The high efficiency achieved by the new design is far beyond the reach of traditional design. Another successful example is using evolutionary algorithms to design antennas for NASA's Space Technology spacecraft [64], and one of the antennas was used in the mission. The antenna is a critical device for a spacecraft to communicate with the ground, as faulty communication may cause a loss in data or even the crash of the spacecraft. The antennas designed using evolutionary algorithms are significantly better than those designed by human experts.

In the following sections, we will focus on three main fields in which evolutionary computation plays a role.

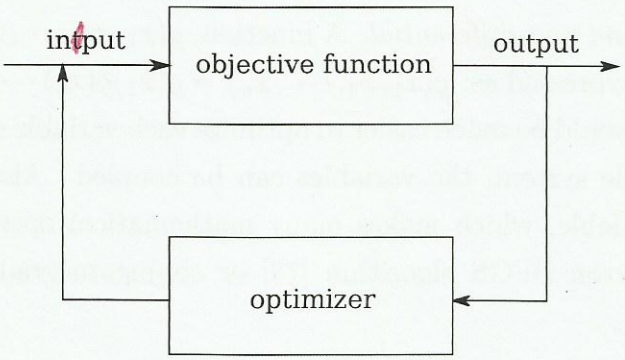


Figure 2.3: This diagram shows the process of black box optimization. The task is to find a candidate solution (input) that can optimize (e.g., maximize or minimize) the objective function.

2.2.3.1 Black Box Optimization

A prominent application of evolutionary computation is black box optimization. Black box optimization refers to optimizing an objective function without assuming the hidden structure of that function (e.g., linear or differentiable) [69]. In particular, the aim is to find a set of inputs that can maximize or minimize the output of the objective function. Fig. 2.3 shows a diagram of the black box optimization. In the following, we list a few cases that evolutionary computation could be applied to solve hard optimization problems [70].

- *High-dimensional:* As the dimension, n, of the objective function increases, the search space increases exponentially. This is called "curve of dimension" by Bellman [71]. If we have to optimize a function that has 30 dimensions, and each dimension only has 20 parameter values to be selected, for a grid search in which all the possible solutions are evaluated, it will take 20^{30} evaluations. Suppose that each evaluation takes $1\mu s$, the grid search would take more than $3 \cdot 10^{31}$ years. However, if using evolutionary computation techniques, it probably takes hours to find the good solution.

- *Multi-Model:* Multi-model means a system (function) has more than one optima (e.g., Rastrigin Function). The one(s) with the best fitness value is (are) considered as global optima, and the others are considered as local optima. These local optima can be misleading for gradient-based search algorithms (e.g., *hill climbing* algorithm), as the solutions may get trapped. Evolutionary algorithms are shown to be effective to find global optima [72].

- *Non-separable and non-differential:* A function, $g(x_1, x_2, \cdots, x_n)$ is non-separable, if it can not be expressed as: $g(x_1, x_2, \cdots, x_n) = g(x_1)g(x_2)\cdots g(x_n)$. For a separable function, it would be much easier to optimize each variable separately. However, for non-separable system, the variables can be coupled. Also, the function may not be differentiable, which makes many mathematical optimization algorithms (e.g., quasi-Newton BFGS algorithm [73] or conjugate gradient algorithm [74]) infeasible.

- *Multi-objective:* When encountering real-world problems, we usually need to deal with multiple objectives, which need to be optimized simultaneously. For example, when designing a car, the engineers need to consider the shape, performance of the engines and cost. As some of the objectives (such as performance and cost) are conflicting, the designer should find a Pareto optimal solution [75], in which none of the values of the objective functions can be increased without decreasing the value of other objective functions. Evolutionary multi-objective optimization is a technique to generate Pareto optimal solutions. For a review, see [75].

2.2.3.2 Evolutionary Robotics

Another application of evolutionary computation is evolutionary robotics, in which the controller and/or morphology of a robot are automatically generated. The user considers the robot as a whole and only needs to specify the optimization criteria which are represented by a fitness function. The fitness function could be single-objective or multi-objective. The aim is to optimize (minimize or maximize) the fitness using evolutionary algorithms.

In evolutionary robotics, the typical problem is to optimize controllers of a robot. The normal procedure is as follows. First, an initial population of random controllers is generated. Each controller is represented by a chromosome. If the controller is a neural network, the chromosome could be a vector of real numbers. Each controller is tested on the robot (in simulation or reality), and the robot's performance for some specific task is measured and evaluated using a pre-defined fitness function. The 'fitter' robot controller (with higher performance) has a higher chance of being selected and generating offspring in the next generation. This process iterates until a good solution is found.

A common control structure adapted in evolutionary robotics is the artificial neural network due to its simple representation [76, 77]. According to the types of control strate-

gies, we can choose different neural networks (forward neural networks and recurrent neural networks, etc.). After fixing the structure of the robot controller, evolutionary algorithms are used for optimizing the weights of the neural networks. One can even evolve the structure (topology) and weights of the neural networks [78].

There are two main research aims in evolutionary robotics: 1) engineering—developing the control strategy for robots; 2) biology—to understand the biological systems using simulated (or physical) evolution. In engineering contexts, a range of work has been presented, ranging from simple behaviors, such as phototaxis behavior [79] to complex behaviors such as navigation [80] and locomotion [81] of robots with multiple degrees of freedom. In biology contexts, evolutionary robotics is used as a tool to understand the general principles of evolution. It provides an efficient way to validate or even create hypothesis based on evolution in simulation or real robots, compared with the slow evolutionary process in nature. AVIDA [82] and AEvol [83] are two computer software systems that are used for studying the evolution of bacterias. In hypothesis validation, evolutionary robotics was used as a tool to investigate some key issues in biology. For example, whether altruism plays an important role in cooperation among species [84, 85], the conditions of emergence of communication during the evolution [86], how morphology and control are coupled [87], and the coevolution of predator and prey [88, 89].

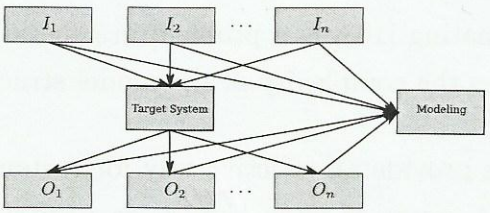
Two methodologies are adapted in evolutionary robotics. The first one is evolving the robot's solutions in simulation and then transferring the best solution to a physical robot. The other is evolving the solutions directly on the physical robot. A drawback of the first method is the *reality gap* [90]. That is, the simulation used for generating the solutions may not match the robot's real operating environment, causing a reduction in performance when testing the solution in reality. Many studies aimed at reducing the *reality gap* [91, 92, 93]. In [91, 92], the authors presented the transferability approach to improve the control quality of a robot. The controllers were generated in simulation, and the best controller (selected according to multiple criteria) was transferred to the real robot, and the data collected were used for refining the simulator. This increased the quality of controllers generated in simulation, and also reduced the number of experiments to be conducted on the real robot. In [80], the evolution was performed directly on the real (Khepera) robot to evolve the controller to perform a navigation task. An advantage of this is that the *reality gap* has been avoided. However, it may take a significant amount of time to evolve a desirable solution. In [80], it took two weeks to evaluate only 100 generations. The power supply is a problem when the robot uses a

battery, as it can only last a limited time. Although the authors in [80] used a wire to connect the robot with a charging station, this is not desirable for outdoor experiments and when using multiple robots. Recently, a distributed online onboard evolutionary method (artificial embodied evolution) has received much attention [94, 95, 96]. In artificial embodied evolution, the population is distributed among different robots, and the gene exchange is done through *mating*. Each robot only exchanges its genes with its nearby neighbors. There is no central control over the group of robots. This approach is suitable for the situation where the environment that the robots are operating in is not predictable or changing after deploying the robots. The robots need to adapt to the changing environment, while satisfying certain basic requirements inserted by the designers. A more challenging research area could be evolving both the morphology and controller of the robots in a distributed manner—*evolution of things* [53]. This forms an open-end evolution among different artifacts, which is similar to the process of how living creatures evolve in real world. The fast development of 3D printing makes this method appealing [97].

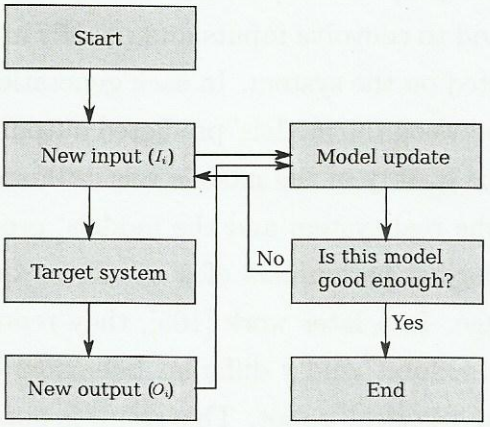
2.2.3.3 System Identification

System identification concerns the synthesis of models of a hidden system through conducting a set of experiments. It is widely used in both academia and industry [98, 11]. The experiments are conducted by feeding a series of inputs into the system and collecting the outputs corresponding to these inputs. The objective is to find a model that fits the inputs and outputs.

There are two system identification approaches: the offline approach and the online approach [98]. In the offline approach, the observed data is collected first, and the aim of modeling is to generate a model that fits the observed data. This method is often used when the data is simple to collect or the parameters and operating environment of the system do not change much in a short time. However, in some cases, the parameters of the system are changing. That means the obtained model based on the previous observation data can not be applied to the new situation any more. Through the online system identification method, the model can be updated using the new observed experimental data. The two approaches of system identification are shown in Fig. 2.4. Note that for some systems, there are only outputs.



(a) offline



(b) online

Figure 2.4: Diagrams showing two approaches for system identification. (I_i, O_i), where $i \in \{1, 2, \cdots, n\}$, represent pairs of input and output data. Redrawn from [99]

System identification can be divided into two main procedures: modeling and estimation. Modeling defines the order or general structure of the hidden system [100]. Estimation identifies the parameters associated with the given structure. There are many algorithms to determine the parameters for a given structure. Typical estimation algorithms are recursive prediction error methods [101], which are based on gradient search. Gradient search methods, such as greedy algorithms, can easily get trapped in local optima, especially when there are numerous local optima near the global optimum. An alternative search method is using evolutionary algorithms. There are some system identification methods that combine the modeling and estimation process, for example, the NARMAX method [11]. Neural networks are also a good representation of the system under investigation, as their structure and weights (parameters) can be optimized simultaneously. The disadvantage of neural networks for modeling is they are difficult to interpret especially when consisting of multiple layers. Genetic programming provides an alternative way for finding the structure and parameters of the system. The models represented by genetic programming can be described in a tree-based structure. As the structure in genetic programming is evolving, the obtained model may be of a different structure

in different runs [102]. Bloating [103] is a problem in genetic programming, where the growth of the tree increases the complexity of the model structure.

Coevolutionary algorithms provide an effective way for system identification [55], [104, 105, 106, 15, 107, 108]. A range of work ~~have~~ has been performed on simulated agents. In [104], Bongard and Lipson proposed the *estimation-exploration algorithm*, a nonlinear system identification method to coevolve inputs and models in a way that minimizes the number of inputs to be tested on the system. In each generation, the input that leads to the highest disagreement between the models' predicted output in simulation was carried out on the real system. The quality of the models was evaluated ~~through~~ by quantitatively comparing the output of the real system and the models' prediction. The method was applied to evolve morphological parameters of a simulated quadrupedal robot after it undergoes 'physical' damage. In a later work [105], they reported that "in many cases the simulated robot would exhibit wildly different behaviors even when it very closely approximated the damaged 'physical' robot. This result is not surprising due to the fact that the robot is a highly coupled, non-linear system: thus similar initial conditions [...] are expected to rapidly diverge in behavior over time". They addressed this problem by using a more refined comparison metric reported in [105]. In [106], an algorithm ~~which~~ that is also based on coevolution of models and inputs was presented to model the simulated quadrotor helicopter and improve the control quality. The inputs were selected based on multiobjective performances (e.g., disagreement ability of models as in [104] and control quality of a given task). Models were then refined through comparing their prediction to each selected test trajectory. In [109], the damage detection process is conducted by a coevolutionary algorithm to extract the maximum information from the system. In [110], a coevolutionary algorithm ~~is~~ was applied to estimate chaotic time series, in which the test data that can extract hidden information from the chaotic system co-evolves with the models. As in other system identification methods, predefined metrics play a critical role for evaluating the performance of models.

Is this a reference?

Many studies also investigated the implementation of evolution directly in physical environments, on either a single robot [111, 112, 33] or multiple robots [113]. In [111], a four-legged robot was built to study how it can infer its own morphology through a process of continuous self-modeling. The robot ran a coevolutionary algorithm onboard. One population evolved models for the robot's morphology, while the other evolved actions (inputs) to be conducted on the robot for gauging the quality of these models through comparing sensor data collected. In [113], a distributed coevolutionary

approach was presented to coevolve on-board simulators and controllers of a swarm of ten robots to perform foraging behavior. Each robot has its own simulator which models the environment. The evolution of each robot's simulator was driven by comparing the real-world foraging efficiency (a pre-defined fitness metric) of its nearby neighbors each executing the best controller generated by their own simulators. Each robot has a population of controllers, which evolved according to the robot's on-board simulator. The best controller was chosen for performing real-world foraging. In all of the above approaches, the model optimization is based on pre-defined metrics (explicit or implicit), which are task dependent.

2.3 Combining AI/Robotics and Animal Behavior

The variety of animal behaviors in nature is immense, ranging from simple perception of light to complicated behaviors such as navigation and communication. The scientific study of animal behavior is pursued not only because it is a subject of interest in itself, but also because the knowledge gained from it has several practical applications. In AI/Robotics, there is a large interest ~~of~~ in studying animal behavior, as the model/knowledge learned can be used to build ~~a~~ more intelligent machines. At the same time, building a machine that mimics ~~the~~ an animal helps researchers better understand its behavior. In this section, we review how AI/robotics and animal behavior study benefit each other. In Section 2.3.1, we introduce some interesting animal behaviors observed in nature. In Section 2.3.2, we detail how animal behavior can be used as inspiration for solving engineering tasks, especially in the area of AI/robotics. In Section 2.3.3, we show how AI/Robotics can contribute to ethology, which is the ultimate goal of this thesis.

2.3.1 Animal Behavior in Nature

Compared with the behaviors exhibited by complex animals such as mammals, insect behavior has also widely studied by researchers. One reason is due to the simple neural system of the insects. This simplicity also makes it more feasible to replicate the intelligence exhibited by the insects.

A basic insect behavior is taxis, which is its intrinsic behavioral response to a specific stimulus. Taxis is divided into different types according to the stimulus which elicits the

insect's response. These behaviors include phototaxis (light), chemotaxis (chemicals), thermotaxis (temperature), etc. For example, a lobster follows the salt-water plume—a kind of chemical signal to find the source. Another interesting taxis behavior of crickets is that the female crickets perform complex auditory orientation behavior towards the male crickets. Researchers have found this complex sound localization behavior emerges from simple reactive steering responses to specific sound pulses generated by male crickets [114]. Apart from taxis behaviors, some insects use the stimuli as cues for navigation or migration. For instance, the bee uses its vision system to navigate in the air and avoid obstacles. Another interesting behavior found in insects is the ball movement of dung beetles [115]. Once the dung beetles form the pieces of dung into a ball, they always roll the dung-ball in a straight line using various stimuli (e.g., the moon, sun and polarised light [116, 117]) as visual cues to transport the food source. This behavior ensures that they keep away from the competitors as far as possible. The stimulus-response behaviors in insects mentioned above are investigated by biologists for centuries. Although the behavior exhibited by insects may be simple and easy to mimic, it is not trivial to be modeled and well understood [118]. When investigating such behaviors, biologists need to learn how to interact with the animal in a meaningful way to extract all of its behavioral repertoire.

Apart from the behavior of a single animal, swarm behaviors, which are emergent (collective) behaviors that arise from the interactions of a number of animals (especially social insects) in a group, have also been widely observed in nature. The individual behaviors in a swarm tend to be relatively simple [17]. The global behavior that is exhibited in a swarm is a result of a self-organized process. Researchers found that individuals do not need the representation or complex knowledge to build a map of what the global behavior should be [119]. There are no leaders in the group. From the point of control, swarm behavior is a distributed control system which does not rely on central coordination.

Many swarm behaviors are observed in nature. For example, the birds' flocking behavior is of particular interest to humans. This is not only because of their beautiful shape formed in a group, but also because of the way the birds coordinate with each other to maintain that shape. A simple mathematical model was proposed by [120] to describe the individual behavior of each bird in a flock. The three rules are: attraction, repulsion and alignment. In the attraction rule, the birds will be attracted by their neighbors, and this would result in each bird moving towards to the 'center' of their neighbors. Repulsion means the bird needs to avoid colliding with each other. Alignment assumes that each

bird moves in the same direction with its neighbors. Although there is no proof that flocking birds follow exactly the three rules, it is attractive that the boids in simulation following such simple rules can mimic the real flocking behavior closely. There are many other swarm behaviors which are also studied extensively such as the aggregation of cockroaches [121], foraging in ants [122], flashing synchronization in fireflies [123], mound building in termites [124].

2.3.2 Swarm Optimization and Swarm Robotics

In the previous sections, we introduce some interesting animal behaviors observed in nature. Many algorithms are inspired from observation of animal behaviors. In this section, we will review two bio-inspired algorithms—ant colony optimization algorithm [125] and particle swarm optimization algorithm [126]. Swarm robotics, which uses the swarm behavioral rules of social insects as an design principle to solve complex tasks, has been paid much attention in recent decades.

2.3.2.1 Swarm Optimization

Ant Colony Optimization

Ant Colony Optimization (ACO) was inspired by the foraging behavior of ants [127]. When an ant finds an item of food, it will leave pheromone on its path back to the nest. Other ants will be attracted by the pheromone, the strength of which may represent the quantity/quality of the food. Researchers have found that this indirect communication, which is know as *stigmergy* [128], leads them to find the shortest path between their nest and location of food source. The initial application of ACO is to find the optimal path in a combinational (discrete) problem. Note that nowadays the application of ACO algorithms ranges from discrete optimization (e.g., routing and load balancing [129]) to continuous optimization [130].

The principle of ACO algorithms can be summarized by the two steps as follows:

- Use the pheromone model (and heuristic model) to generate candidate solutions. The pheromone model is a parameterized probability distribution in the search space. The heuristic model could be the length of the segment chosen.

- The candidate solutions are used as a bias for sampling high-quality solutions in the next iteration.

For a complete implementation of ACO algorithms, see [127].

Particle Swarm Optimization

Particle Swarm Optimization (PSO) is another optimization algorithm which is inspired by the flocking of birds and schooling of fish. It was first proposed by Kennedy and Eberhart [126]. The initial application is to optimize the weights of neural networks—a continuous optimization problem. It is also widely used in discrete optimization.

The basic component in PSO is called *particle*. A PSO algorithm consists of a finite set of particles. The movement of each particle is updated using *velocity*. The velocity of each particle in each time step is updated based on its current velocity, the deviation between the best position (the particle has found so far) and its current position, and the deviation between the best position by its neighbors and its current position. This usually results in the particles moving towards the high-quality solutions after certain iterations. The update of each particle can be written using two equations as follows [126]:

$$\overrightarrow{v}_{i+1} = \overrightarrow{v_i} + c_1\overrightarrow{R_1} \otimes (\overrightarrow{p_i} - \overrightarrow{x_i}) + c_2\overrightarrow{R_2} \otimes (\overrightarrow{p_g} - \overrightarrow{x_i}). \tag{2.4}$$

$$\overrightarrow{x}_{i+1} = \overrightarrow{x_i} + \overrightarrow{v_i}. \tag{2.5}$$

$\overrightarrow{R_1}$ and $\overrightarrow{R_2}$ are independent random number generators that return a vector of random values in range $[0, 1]$. c_1 and c_2 are referred to as acceleration coefficients. $(\overrightarrow{p_i} - \overrightarrow{x_i})$ and $(\overrightarrow{p_g} - \overrightarrow{x_i})$ represent respectively the deviation between the best position (it has found so far) and its current position, and deviation between the best position by its neighbors and its current position. The first item in Eq. (2.4) keeps the particle moving in the previous direction; the second item makes the particle move towards the best position of its own; the third position forces the particle move towards to the best position that its neighbors have found. Eq. (2.5) updates the particle's position.

2.3.2.2 Swarm Robotics

In the previous section, we described how swarm behaviors have inspired the design of novel optimization algorithms. In this section, we introduce how to apply swarm intelligence techniques to multi-robot research, which is referred to as swarm robotics. The behavior of many social insects (e.g, ants, termites, wraps and bees) has been used as inspirations in swarm robotics.

Swarm robotics investigates how multiple robots each with limited ability communicate, coordinate and self-organize to accomplish complex tasks. The advantages of swarm robotics are as follows:

- *Robustness:* The robustness of a swarm robotic system can be explained in the following: 1) if some robots fail, the other robots could replace the functions of the failed robots; 2) the control is distributed; 3) as the individual is simple, it has fewer components that could get damaged. Note that swarm robotic systems can also be affected by errors. That is, some individuals' failure would influence the whole self-organized process [131].

- *Scalability:* In swarm robotics, the exact number of robots does not have significant impact on the global performance/behavior of the system (unless the number is sufficiently small). That is, increasing/decreasing the number of robots, the system should be still under control and the coordination is still maintained. When investigating the performance of a swarm robotic system, a scalability study is usually considered [132, 133].

- *Flexibility:* The swarm could easily adapt to the changing tasks and generate relevant solutions [134]. The role of each robot in the swarm could be changed depending on the need for the task.

In order to coordinate, the robots need to interact with each other and environment. There are three kinds of interaction in swarm robotic systems [135].

- *Interaction via environment:* In this interaction method, the robots communicate with each through changing the environment. There is no explicit communication between each robot. In nature, the pheromone ants leave when foraging is an environmental stimulus for locating ~~the~~ a food source.

- *Interaction via perception:* In this method, the robots can perceive each other in a limited range. This perception is local and there is no explicit communication between the robots. The robots can distinguish different items (e.g., other robots, objects) in the environment. In nature, when ants need to collectively pull food to the nest, they need to perceive each other (to avoid collision) and the food (object).

- *Interaction via explicit communication:* This type of communication could be realized by broadcast (e.g., WiFi [136]) or a distributed sensing network [137]. How to build a reliable network when the number of robots is significantly large is still a topic widely discussed. When the number of robots increases, the load of communication increases exponentially. A possible solution is combining the advantage of network communication and local communication using the robots' perception.

A range of tasks have been demonstrated in swarm robotics. The tasks range from aggregation [138, 21, 139, 121], dispersion [140, 141], pattern formation [142, 143], collective movement [144] to cooperative transport [145, 146, 147, 132], etc. Aggregation can be considered as the fundamental behavior of other more complex tasks. In [121], a group of robots mimic the cockroaches' aggregation behaviors, in which the robots join or leave a nest (place) in the environment with a probability proportional to the size of the nest. In [21], the robots each with a binary sensor were reported to aggregate into a single cluster, validated using 40 e-puck robots. The robots did not perform algorithmic computation. The aggregation performance scaled well with 1000 robots in simulation. In [148], Werfel et al. designed a group of termite-inspired robots that work collectively to build several structures. The robots communicate with each other using *stigmergy*. In [144], a group of nine Kobot robots were reported to mimic the flocking behavior of birds. These robots followed some simple rules similar to those proposed by Reynolds [120]. In cooperative transport, Chen et al. [132] presented a strategy for a group of miniature robots to transport a tall object to the goal. In the task, the robots only push the object when the robots' vision of the goal is occluded by the object. This strategy was proved to be able to push any convex object to the goal in a 2D environment.

2.3.3 Contribution of AI/Robotics to Ethology

In the previous sections, it was reviewed how the study of animal behavior study can be used as inspiration for designing optimization algorithms and robotic systems. However, AI/robotics can also benefit the study of animal behavior. This section reviews two approaches: *learning from synthesis* and *robot-animal interaction.*

2.3.3.1 Learning from Synthesis

Ethologists have studied animal behavior over a century. There are some basic steps that ethologists follow in the study of animal behavior [149]. The first step is observation, and after that they formulate some scientific questions on the observed behavior, and generate hypothesis to answer these questions. They then conduct related experiments on the animals and collect data. After analyzing the data, the conclusion will be made to support or reject their hypothesis.

Robotics or artificial life can be used as an alternative methodology to investigate and understand animal behavior. Robots can be used as physical models of animal behaviors for testing hypotheses [150, 6]. For example, taxis behavior has often been implemented on mobile robotic systems for investigating steering and navigation [118]. In [151], Webb used a robot to model the phonotaxis behavior of crickets [152]. The robot can locate the position of a sound source and move towards it under different conditions. There was a good agreement between data collected from experiments on the robot and the animal. Another taxis behavior—chemotaxis, in which animals follow a specific chemical trail, has been used as a model for robots to find odor sources based on artificial neural networks [153] and even Braitenberg vehicles [154]. Robots can be used as a validation for the models obtained from biologists and allow them to better understand the animal behavior from a synthetic point of view. Besides, roboticists can generate new hypotheses and test them using (simulated or physical) robots. In social behavior study, Balch et al. [155] built executable models of the behaviors ~~and~~ of ants and monkeys, which can be directly executed by multi-robot systems. The aim is to show how research into multi-robot systems can contribute to the study of collective animal behaviors. In [156], Chappell et al. argue that there are many ways in which biologists interested in natural intelligence can learn from AI and robotics, and they outline specific kinds of contributions that AI can make to biological ~~study~~ studies. They also give some suggestions on

how AI and robotics researchers could collaborate with biologists to solve some cutting-edge problems in animal behavior.

As opposed to the works mentioned above, the method proposed in this thesis aims to synthesize models of agent (animal) behaviors automatically, rather than manually. This could help to spare scientists from having to perform numerous laborious experiments, allowing them instead to focus on using the generated models to produce new hypotheses.

2.3.3.2 Robot–Animal Interaction

Besides pure robot-based or AI research, researchers also use robots to interact with real animals. They build and program robots (i.e., replicas) that can be inserted into the group of social animals [157, 158, 159, 160, 161]. Robots can be created and systematically controlled in such a way that they are accepted as con- or hetero-specifics by the animals in the group [162]. In this case, one "animal" in a group is completely controlled and they can observe the behaviors of the mixed society [163]. The behavior of the inserted robot can be controlled and the model can also be embedded into the robot for verification [164]. The behavior of robots can be programmed in such a way that its behavior is not influenced by the other real animals in the group, and they can be used as demonstrators or leaders in the experiments. Further more, it is simpler to test a hypothesis through controlled interaction in social behaviors.

In [157], a replica fish which resembled the appearance (i.e., visual morphology) of sticklebacks was created to investigate two types of interaction: recruitment and leadership. In [165], a robot-fish that can interact intelligently with live zebrafish to study their preference and locomotion behavior was designed. In [159], autonomous robots which executed a derived model were mixed into a group of cockroaches to modulate their decision-making of selecting shelter in the aggregation behavior. The robots behaved in a similar way to the cockroaches. Although the robots' appearance was different to that of the cockroaches, the robots released a specific odor that the cockroaches could detect and regard the robots as conspecifics. In [166, 161], Vaughan et al. have built a mobile robot that can interact with ducks in a circular arena and drive them to the safe place. In [167], Gribovskiy et al. designed a robot ~~which is~~ capable of interacting with chicks to study how the behavior of chicks is influenced by the others in a group.

Although robots which are well designed can be mixed ~~in~~ with social animals, building such kinds of robots is a time-consuming process. It is also expensive to some extent and

requires the collaboration of researchers from different disciplines. In the aforementioned works, the models were manually derived and the robots were only used for model validation. We believe that this robot-animal interaction framework could be enhanced through the system identification method proposed in this dissertation, which autonomously infers the collective behavior.

↑
dissertation or thesis?